A Systolic Array Implementation of Discrete Relaxation Algorithm

UUCS-TR-86-008

Wei Wang*, Jun Gu, and Thomas C. Henderson

**Department of Electrical Engineering*
Department of Computer Science

University of Utah
Salt Lake City, UT 84112

12 March, 1986

Abstract

Discrete Relaxation techniques have proven useful in solving a wide range of problems in digital signal processing, artificial intelligence, machine vision, and VLSI engineering, etc. A conventional hardware design for an 8-label 8-object Discrete Relaxation Algorithm (DRA) requires three 4K memory blocks and the maximum execution time of over seconds and minutes, which makes such a DRA hardware implementation infeasible. A highly parallel systolic array for the computation of an 8-label 8-object DRA problem has been developed. This realization eliminates the 12K memory requirement and performs DRA computation at the worst case in microseconds. The circuit requires about 6,382 transistors. Major design issues and chip descriptions are described in this paper.

This work is supported in part by NSF Grants MCS-82-21750, DCR-85-06393, and DMC-85-02115; and in part by a University of Utah Research Fellowship.

Table of Contents

List of Figures

1.Introduction and Motivation

Relaxation is a very general computational technique for a wide range of theoretical and engineering problems. Since its invention many years ago it has demonstrated powerful and extensive applications in many areas. Some of them are listed below:

1. ***Digital Signal Processing***: such as digital signal and digital image filtering;

2. ***Mathematics***: (1) for solving linear and certain partial differential equations; (2) finding out the objective function; (3) performing linear programming and optimization;

3. ***Artificial Intelligence***: doing heuristic optimal search and propagating numeric constraints, etc.;

4. ***VLSI Engineering***: for building various kinds of relaxation simulation tools and for developing a hardware accelerator;

5. ***Computer Vision***: dealing with the problems such as graph homomorphism, graph coloring and image understanding. For line finding, stereopsis, line-labeling, and semantics-based region growing, etc.;

6. ***Robotics***: for solving its vision problems;

7. ***Mechanics***: in computing stresses,etc.

For a review of the numerous applications of relaxation processes see [11,12,13,14,15,16].

Classical relaxation *(CR)* was introduced by Southwell in 1940 [11] and the symbolic (as opposed to numeric) versions of relaxation *(SR)* were introduced in the mid-seventies [12]. The version used here is that described by Henderson [Note on Disc]. The ***Discrete Relaxation Algorithm (DRA)*** is a restriction of the classical relaxation process to systems of Boolean inequalities which take values over the two element set {0,1}. One of the significant technique resulting from the introduction of DRA is that both classical and symbolic relaxation algorithms are directly executable in silicon subroutines, thus making many real-time relaxation applications feasible. The project described in this paper is the first successful hardware implementation of this algorithm.

2.Discrete Relaxation Algorithm (DRA)

2.1 Boolean Formulation of Discrete Relaxation

Instead of seeking a real number solution in a numerical relaxation situation [13], the solution to be found in discrete relaxation case involves the assignment of a set of *labels* at each unknown such that some constraint relation among the labels is satisfied by neighboring unknowns. Whereas the unknowns in numerical relaxation take on real number values, the unknowns in a *labeling problem* take on a Boolean vector value with each element in the vector corresponding to a possible label. Boolean vector operations are denoted by ', ×, t, *, + and · which represent complementation, vector multiplication, transpose, Boolean "and," Boolean "or," and Boolean vector dot product, respectively. Let

1. $U = \{u_1,...,u_n\}$ be the set of unknowns,
2. $\Lambda = \{\lambda_1,...,\lambda_m\}$ be the set of possible labels,
3. $\Lambda_i = (l_1,...,l_m)^t$ be the column vector describing the set of labels (i.e., zero or one) possible for u_i, where l_j=1 if λ_j is compatible with u_i; 0 otherwise.
4. **C** be an m by m compatibility matrix for label pairs, where **C**(i,j)=1 if λ_i is compatible with λ_j; 0 otherwise.
5. $\Lambda_{ij} = (\Lambda_i \times \Lambda_j^t)*((Nei(i,j)'E+C)$ be an m by m compatibility matrix for u_i and u_j, where E is the m by m matrix for all 1's, and Nei(i,j) = 1 if u_i neighbors u_j; 0 otherwise.
6. Λ_k denotes k^{th} row of Λ_{ij}.

A *labeling* is a vector $L = (L_1,...,L_n)^t$, where $L_i = (l_{i1},...,l_{i'm})$ in Λ_i is a Boolean vector with l_{ij} = 1 if label λ_j is a possible label for object u_i; 0 otherwise. A labeling is *consistent* if for every i and k:

$$l_{ik} \leq \prod_{j=1}^{n} [\sum_{p=1}^{m} (l_{ik} \times l_{jp} \times \Lambda_{ij}(k,p))] \qquad (1)$$

It can be rewritten as:

$$l_{ik} \leq l_{ik} \times \prod_{j=1}^{n} [\sum_{p=1}^{m} (l_{jp} \times \Lambda_{ij}(k,p))] \qquad (2)$$

If the l_{ik}'s, k=1,m are now gathered together in vector form:

$$\begin{bmatrix} l_{i1} \\ l_{i2} \\ \vdots \\ l_{im} \end{bmatrix} \leq \begin{bmatrix} l_{i1} \\ l_{i2} \\ \vdots \\ l_{im} \end{bmatrix} * \begin{bmatrix} \sum_{p=1}^{m} (l_{1p} * \Lambda_{i1}(1,p)) \\ \sum_{p=1}^{m} (l_{1p} * \Lambda_{i1}(2,p)) \\ \vdots \\ \sum_{p=1}^{m} (l_{1p} * \Lambda_{i1}(m,p)) \end{bmatrix} * \ldots * \begin{bmatrix} \sum_{p=1}^{m} (l_{np} * \Lambda_{in}(1,p)) \\ \sum_{p=1}^{m} (l_{np} * \Lambda_{in}(2,p)) \\ \vdots \\ \sum_{p=1}^{m} (l_{np} * \Lambda_{in}(m,p)) \end{bmatrix} \quad (3)$$

or

$$\begin{bmatrix} l_{i1} \\ l_{i2} \\ \vdots \\ l_{im} \end{bmatrix} \leq \begin{bmatrix} l_{i1} \\ l_{i2} \\ \vdots \\ l_{im} \end{bmatrix} * \begin{bmatrix} L_1 * \Lambda_{i1}(1)^t \\ L_1 * \Lambda_{i1}(2)^t \\ \vdots \\ L_1 * \Lambda_{i1}(m)^t \end{bmatrix} * \ldots * \begin{bmatrix} L_n * \Lambda_{1n}(1)^t \\ L_n * \Lambda_{1n}(2)^t \\ \vdots \\ L_n * \Lambda_{1n}(m)^t \end{bmatrix} \quad (4)$$

or

$$L_i \leq L_i * \prod_{j=1}^{n} \left(\left\{ \left[L_j \right] \times \left[\Lambda_{ij}(1)^t \cdots \Lambda_{ij}(m)^t \right] \right\}^t \right). \quad (5)$$

Let

$$P = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ \vdots \\ P_n \end{bmatrix}, \quad (6)$$

where the column vector:

$$P_i = \prod_{j=1}^{n} (\{[L_j] \times [\Lambda_{ij}(1) \ldots \Lambda_{ij}(m)]\}^t). \quad (7)$$

Gathering together the L_i's, i=1,n, we have

$$L \leq L * P \quad (8)$$

This formulation emphasizes the relation to classical relaxation. The relaxation is achieved by repeating

$$L \Leftarrow L * P \quad (9)$$

until L does not change value.

2.2 An Example

Suppose that we are analyzing a picture of a scene, with the aim of describing it, and that we have detected a set of objects $u_1,...,u_n$ in the scene, but have not identified them unambiguously. The relationships that exist among the objects are used to eliminate the ambiguity.

An example for eliminating the ambiguity in a region coloring problem is given here to demonstrate these ideas and computation procedures. For simplicity, consider the case of three regions to be colored red, green or blue with the constraints:

1. Region 1 must be red.
2. Region 3 must be blue, and
3. No two regions may be colored the same color.

Thus, u_i = Region i (for i=1,2,3) and:

$$U = \{u_1, u_2, u_3\}$$

$$\Lambda=\{\lambda_1, \lambda_2, \lambda_3\}$$

where λ_1 is red, λ_2 is green, and λ_3 is blue. Since region 1 must be red, we have:

$$\Lambda_1=[1\ 0\ 0]^t$$

and since region 3 must be blue:

$$\Lambda_3=[0\ 0\ 1]^t$$

Finally, since there is no restriction on region 2's color, we have all possibilities:

$$\Lambda_2=[1\ 1\ 1]^t$$

Since only similar colors are incompatible, we have:

$$C=\begin{pmatrix} 0\ 1\ 1 \\ 1\ 0\ 1 \\ 1\ 1\ 0 \end{pmatrix} \qquad (10)$$

for different objects, and

$$C=\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix} \tag{11}$$

for the same object.

We see then that C actually depends on the objects under consideration; i.e., technically, we should write C_{ij} which is identified as:

$$C_{ij}=\begin{pmatrix}Nei(i,j)' & Nei(i,j) & Nei(i,j)\\ Nei(i,j) & Nei(i,j)' & Nei(i,j)\\ Nei(i,j) & Nei(i,j) & Nei(i,j)'\end{pmatrix} \tag{12}$$

where

$$Nei(i,j)=\begin{cases}0 \text{ if Region i does not neighbor Region j,}\\ 1 \text{ if Region i does neighbor Region j.}\end{cases} \tag{13}$$

Now we can calculate Λ_{ij} as:

$$\Lambda_{11}=([1\ 0\ 0]^t\times[1\ 0\ 0])*((0'E)+C)=\begin{pmatrix}1&0&0\\0&0&0\\0&0&0\end{pmatrix}*\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}=\begin{pmatrix}1&0&0\\0&0&0\\0&0&0\end{pmatrix} \tag{14}$$

$$\Lambda_{12}=([1\ 0\ 0]^t\times[1\ 1\ 1])*((1'E)+C)=\begin{pmatrix}1&1&1\\0&0&0\\0&0&0\end{pmatrix}*\begin{pmatrix}0&1&1\\1&0&1\\1&1&0\end{pmatrix}=\begin{pmatrix}0&1&1\\0&0&0\\0&0&0\end{pmatrix} \tag{15}$$

$$\Lambda_{13}=([1\ 0\ 0]^t\times[0\ 0\ 1])*((1'E)+C)=\begin{pmatrix}0&0&1\\0&0&0\\0&0&0\end{pmatrix}*\begin{pmatrix}0&1&1\\1&0&1\\1&1&0\end{pmatrix}=\begin{pmatrix}0&0&1\\0&0&0\\0&0&0\end{pmatrix} \tag{16}$$

$$\Lambda_{21}=([1\ 1\ 1]^t \times [1\ 0\ 0])*((1'E)+C)=\begin{pmatrix}1&0&0\\1&0&0\\1&0&0\end{pmatrix}*\begin{pmatrix}0&1&1\\1&0&1\\1&1&0\end{pmatrix}=\begin{pmatrix}0&0&0\\1&0&0\\1&0&0\end{pmatrix} \quad (17)$$

$$\Lambda_{22}=([1\ 1\ 1]^t \times [1\ 1\ 1])*((0'E)+C)=\begin{pmatrix}1&1&1\\1&1&1\\1&1&1\end{pmatrix}*\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}=\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix} \quad (18)$$

$$\Lambda_{23}=([1\ 1\ 1]^t \times [0\ 0\ 1])*((1'E)+C)=\begin{pmatrix}0&0&1\\0&0&1\\0&0&1\end{pmatrix}*\begin{pmatrix}0&1&1\\1&0&1\\1&1&0\end{pmatrix}=\begin{pmatrix}0&0&1\\0&0&1\\0&0&0\end{pmatrix} \quad (19)$$

$$\Lambda_{31}=([0\ 0\ 1]^t \times [1\ 0\ 0])*((1'E)+C)=\begin{pmatrix}0&0&0\\0&0&0\\1&0&0\end{pmatrix}*\begin{pmatrix}0&1&1\\1&0&1\\1&1&0\end{pmatrix}=\begin{pmatrix}0&0&0\\0&0&0\\1&0&0\end{pmatrix} \quad (20)$$

$$\Lambda_{32}=([0\ 0\ 1]^t \times [1\ 1\ 1])*((1'E)+C)=\begin{pmatrix}0&0&0\\0&0&0\\1&1&1\end{pmatrix}*\begin{pmatrix}0&1&1\\1&0&1\\1&1&0\end{pmatrix}=\begin{pmatrix}0&0&0\\0&0&0\\1&1&0\end{pmatrix} \quad (21)$$

$$\Lambda_{33}=([0\ 0\ 1]^t \times [0\ 0\ 1])*((0'E)+C)=\begin{pmatrix}0&0&0\\0&0&0\\0&0&1\end{pmatrix}*\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}=\begin{pmatrix}0&0&0\\0&0&0\\0&0&1\end{pmatrix} \quad (22)$$

The (p,q) entry of Λ_{ij} tells if λ_p at object i is compatible with λ_q at object j. For example, Λ_{11} reveals that only λ_1 is compatible with λ_1 at object 1; i.e., that Region 1 must be colored red.

Finally we continue the iteration process. According to equation (2),

For i=1 and k=1:

$$\begin{aligned} l_{11}^{(n)} \le\ & l_{11}^{(n-1)}*[l_{11}^{(n-1)}*\Lambda_{11}(1,1)+l_{12}^{(n-1)}*\Lambda_{11}(1,2)+l_{13}^{(n-1)}*\Lambda_{11}(1,3)] \\ & *[l_{21}^{(n-1)}*\Lambda_{12}(1,1)+l_{22}^{*(n-1)}\Lambda_{12}(1,2)+l_{23}^{(n-1)}*\Lambda_{12}(1,3)] \\ & *[l_{31}^{(n-1)}*\Lambda_{13}(1,1)+l_{32}^{(n-1)}*\Lambda_{13}(1,2)+l_{33}^{(n-1)}*\Lambda_{13}(1,3)] \end{aligned} \quad (23)$$

$$\begin{aligned} 1 \le\ & 1*[1*1+0*0+0*0] \\ & *[0*0+0*0+1*1] \\ & *[1*0+1*1+1*1] \end{aligned}$$

$1 \leq 1$

$1 \leq 1$ which is true

This says that the color red is all right for Region 1. To determine if the color red is possible for Region 2, we must find l_{21}.

For i=2 and k=1:

$$\begin{aligned} l_{21}^{(n)} \leq l_{21}^{(n-1)} * [&l_{11}^{(n-1)} * \Lambda_{21}(1,1) + l_{12}^{(n-1)} * \Lambda_{21}(1,2) + l_{13}^{(n-1)} * \Lambda_{21}(1,3)] \\ * [&l_{21}^{(n-1)} * \Lambda_{22}(1,1) + l_{22}^{(n-1)} * \Lambda_{22}(1,2) + l_{23}^{(n-1)} * \Lambda_{22}(1,3)] \\ * [&l_{31}^{(n-1)} * \Lambda_{23}(1,1) + l_{32}^{(n-1)} * \Lambda_{23}(1,2) + l_{33}^{(n-1)} * \Lambda_{23}(1,3)] \end{aligned} \quad (24)$$

$1 \leq 1 * [1*0+0*0+0*0]$
$\quad * [1*1+1*0+1*0]$
$\quad * [0*0+0*0+1*1]$

$1 \leq 1*0$

$1 \leq 0$ which is false.

Thus, l_{21} must be set to zero. Likewise, for i=2 and k=3, l_{23} is set to zero, and blue is not a possible label for Region 2. Finally,

For i=2 and k=2:

$$\begin{aligned} l_{22}^{(n)} \leq l_{22}^{(n-1)} * [&l_{11}^{(n-1)} * \Lambda_{21}(2,1) + l_{12}^{(n-1)} * \Lambda_{21}(2,2) + l_{13}^{(n-1)} * \Lambda_{21}(2,3)] \\ * [&l_{21}^{(n-1)} * \Lambda_{22}(2,1) + l_{22}^{(n-1)} * \Lambda_{22}(2,2) + l_{23}^{(n-1)} * \Lambda_{22}(2,3)] \\ * [&l_{31}^{(n-1)} * \Lambda_{23}(2,1) + l_{32}^{(n-1)} * \Lambda_{23}(2,2) + l_{33}^{(n-1)} * \Lambda_{23}(2,3)] \end{aligned} \quad (25)$$

$1 \leq 1 * [1*1+0*0+0*0]$
$\quad * [1*0+1*1+1*0]$

*[0*0+0*0+1*1]

$1 \leq 1$ which is true.

We see then that the value of l_{11}, l_{22}, and l_{33} are not affected by the change of l_{21} and l_{23} to zero. In fact, the system of equations stabilizes after the change of l_{21} and l_{23}, and the result is $l_{11} = l_{22} = l_{33} = 1$, while all other hypotheses are zero. Thus, the only consistent labeling is to label Regions 1,2 and 3 the colors red, green and blue, respectively.

2.3 The Hardware Implementation Problem for DRA

The problem of ***DRA Hardware Implementation 1 (DRA1)*** has been defined as finding out the *labeling matrix L* (n = m):

$$L=(L_1,L_2,...,L_i,...,L_n)^t=\begin{pmatrix} l_{11}, l_{12},..., l_{1n} \\ l_{21}, l_{22},..., l_{2n} \\ ..... \\ ..... \\ l_{n1}, l_{n2},..., l_{nn} \end{pmatrix} \quad (26)$$

for the predefined computational model provided by the *initial labeling matrix*:

$$\Lambda=(\Lambda_1,\Lambda_2,...,\Lambda_n)^t=\begin{pmatrix} l_{11}^{(o)}, l_{12}^{(o)},..., l_{1n}^{(o)} \\ l_{21}^{(o)}, l_{22}^{(o)},..., l_{2n}^{(o)} \\ ..... \\ ..... \\ l_{n1}^{(o)}, l_{n2}^{(o)},..., l_{nn}^{(o)} \end{pmatrix} \quad (27)$$

and unchanged *label pairs' compatibility matrixes* C_{ii} and C_{ij} in equation (8) for every i and j (i,j = 1, 2,..., n).

3.Implementation of Discrete Relaxation Algorithm Using A Systolic Array

3.1 Complexity Analysis

A conventional design DRA1 for an 8-label 8-object DRA problem is presented in section 5.1 and [2]. The computational strategy used in that design is to serially compute each intermediate element of matrixes $\Lambda_{ij}(p,q)$ and l_{ij} and periodically read and write L, Λ, $\Lambda_{ij}(p,q)$ and C_{ij} from and into memories. Since the computation mechanism imbedded in this design is purely an *I/O bounded* computation, the upper bound of execution time is on the order of hours for an NMOS process. Finally the complete system takes 3 separate chips (totally about 80,000 transistors). This design has revealed the inherent computation complexity for DRA's hardware implementation.

Referring to equations (14) to (22), in order to store the initial labels Λ, matrixes C_{ij}, and the intermediate results of all elements of matrixes $\Lambda_{ij}(p,q)$ (i, j, p, q = 1,...,n), the space complexity is on the order of

$$O(2n^2 + 3n^4) = O(n^4). \tag{28}$$

For practical application, the label number could be 8, 16 or 32, thus the bit memory requirements for these different cases are 12K, 48K and 192K, respectively. As shown in design [2], this has added to the circuit size and which has been a bottleneck when n is large.

The time complexity can be estimated from equations (23) to (25). During each iteration, at least $2 \times 4 \times n^4 \times n$ read and write memory operations will need to be performed. Assuming $t_{read} \approx t_{write} \approx 500$ ns for an NMOS process, the computation time complexity of each iteration is $O(n^5)$ (taking the assumption that the unit time is 500ns). Multiplying the worst case iteration times $O(n^2)$ [12], which is determined by the feature of the computational model, the execution is terribly slow.

3.2 Two Considerations for DRA Hardware Design

Two considerations in designing DRA have become critical and challenging.

1. Concurrency and Communication

It should be clear that any attempt to speed up an I/O-bound computation like design [2] must rely on an increase in the memory bandwidth. Since the technological trend clearly indicates a diminishing growth rate for device speed, any major improvement in computation speed must come from the concurrent use of many processing elements [3,5,6]. The degree of concurrency in a special-purpose system is largely determined by the underlying algorithm. Massive parallelism can be achieved if the algorithm is designed to introduce a high degree of *pipelining* and *multiprocessing*. When a large number of processing elements work simultaneously, coordination and communication become significant - especially with VLSI technology where routing costs dominate the power, time, and area required to implement a computation. The issue here for DRA is to design a hardware algorithm that supports a high degree of concurrency, and in the mean time employs only simple, regular communication and control to enable efficient implementation.

2. Simple and Regular Design

Cost-effective designs have also been a chief concern in designing special-purpose chips like DRA. Special-purpose design costs can be reduced by the use of appropriate architectures. If DRA can truly be decomposed into a few types of simple substructures or building blocks, which are used repetitively with simple interfaces, great savings in

design cost can be achieved. To cope with the circuit design complexity, simple and regular designs, similar to some of the techniques used in constructing large software systems, are essential. In addition, special-purpose systems based on simple, regular designs are likely to be modular and therefore adjustable to various performance goals - that is, system cost can be made proportional to the performance required. This suggests that meeting the architectural challenge for simple, regular, modular designs yields a cost-effective DRA chip.

Systolic system [3,5,6] is an attempt to capture the concepts of parallelism, pipelining, and interconnection structures in a unified framework of mathematics and VLSI engineering. They embody engineering techniques such as multiprocessing and pipelining together with the more theoretical ideas of cellular automata and algorithms, and therefore are excellent ideas for DRA hardware implementation.

3.3 A Highly Parallel Reformulation for DRA

The hardware parallel reformulation of DRA takes the following three steps in order to solve the complexity met in the conventional DRA1 design.

1. Constructing the Parallel Computation Tree

When more effort is spent analyzing Eq. (2), we see that element $\Lambda_{ij}(k,p)$ can be decomposed as

$$\Lambda_{ij}^{(n-1)}(k,p)=l_{ik}^{(o)}l_{jp}^{(o)}c_{k,p} \tag{29}$$

which can form a leaf node like

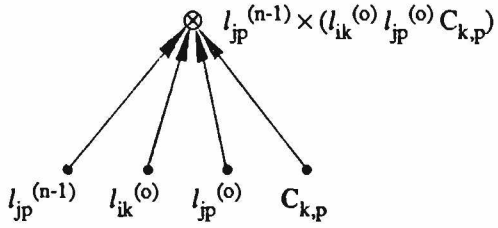


Figure 1: Leaf Node for Computing $l_{jp} \times \Lambda_{ij}(k,p)$

so that Eq. (2) can be hierarchically formed as a tree-like structure with each level imbedded in the parallel computation for their leaves' operands as shown in Figure 3.

2. Speeding Up the Iteration

The node computation in Figure 1 can be speeded up by replacing $l^{(o)}_{ik}$ and $l_{jp}^{(o)}$ with $l_{ik}^{(n-1)}$ and $l_{jp}^{(n-1)}$. The modified computation composed of the leaf node:

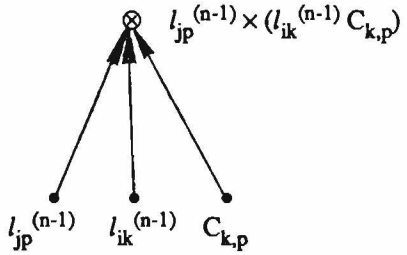


Figure 2: Modified Leaf Node Computation for $l_{jp} \times \Lambda_{ij}(k,p)$

The computation tree for $l^{(n)}_{ik}$ is formed:

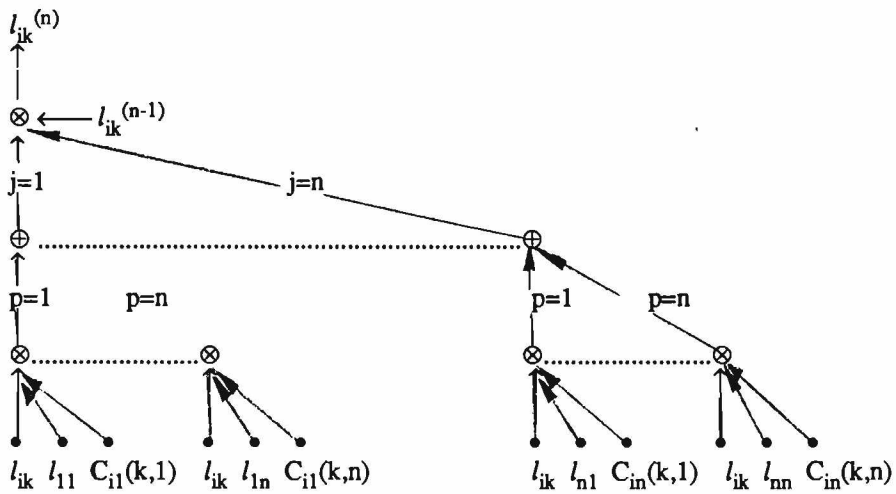


Figure 3: A Parallel Tree Structure for Computing n^{th} l_{ik}

3. Introducing Time Dimension in Computation

To compute an n-label relaxation problem, the total number n^2 of l_{ik}'s needs to be evaluated. This means at least 64 computation trees as shown in Figure 3 need to be built inside the circuit, which greatly increases the circuit size. To minimize this problem, each operand at the bottom of the tree has been constructed in a time-dimension. As the time changes, l_{ik} (i = 1,..., n) can be generated. This computation philosophy does not add more time complexity but decreases the computation tree requirement to n [7]. The introduction of time dimension constitutes the theoretical basis for systolically circulation and interleaved processing.

The hardware parallel reformulation for DRA1 eliminates three 4K memories from the design [2]; only a 64-bit shift register is required to store all 64 intermediate label elements. Thus space complexity is decreased to $O(n^2)$. Since each computation takes 64 cycles, assuming a clock cycle is about 150 ns (NMOS process) and the maximum iteration time is $O(n^2)$, the execution time using this highly parallel computation at the worst case is given in microseconds.

3.4 Basic Principles and Implementations for DRA2 Circuit

1. System Architecture and Block Diagram

1. System Architecture and Block Diagram

The block diagram of the DRA2 circuit is illustrated in Figure 4. The chip consists of four functional blocks.

1. **Compatibility Matrix Registers (CMR). C_{ij} Registers** are a set of eight 8-bit shift registers in the leftmost part of the circuit, they are used for storing each C_{ij} matrix. Another set of **C_{ii} Registers** in the rightmost part of the circuit are for storing C_{ii}.

2. **8 × 8 Systolic Array (SA).** The systolic array is composed of 8 by 8 simple and regular cells. 8 simple and regular cells. They are predefined to map the highly parallel computation algorithm of Figure 3 into silicon. A number of horizontal and vertical communication wires are designed around the four edges of the cells to make use of higher degrees of parallelism in the computation.

3. **L-matrix Shift Register (LSR).** It is used for (1) the input and output data paths for original and final labeling matrices, (2) the pipelining channel for tree-root operands broadcasting and pipelining, forming a recursive DRA computational wavefront and (3) performing temporarily the data storing and updating.

4. **Control Module (CM).** This module includes four units. An **8-Bit Comparator** is located on top of the first 8-bit shift register of the LSR to sense the equality between the n^{th} output vector $L_i^{(n)}$ of the systolic array and the corresponding $n\text{-}1^{th}$ row vector $L_i^{(n-1)}$ inside the LSR. A **Timer** is served as both the systole pacer and tagged-bit signal generator for iteration control. An **8-Bit State Register** is used for collecting comparison results from the Comparator and monitoring iteration states. Finally a **Finite State Machine (FSM)** is built for performing a self-timed synchronization among these functional blocks and host computer.

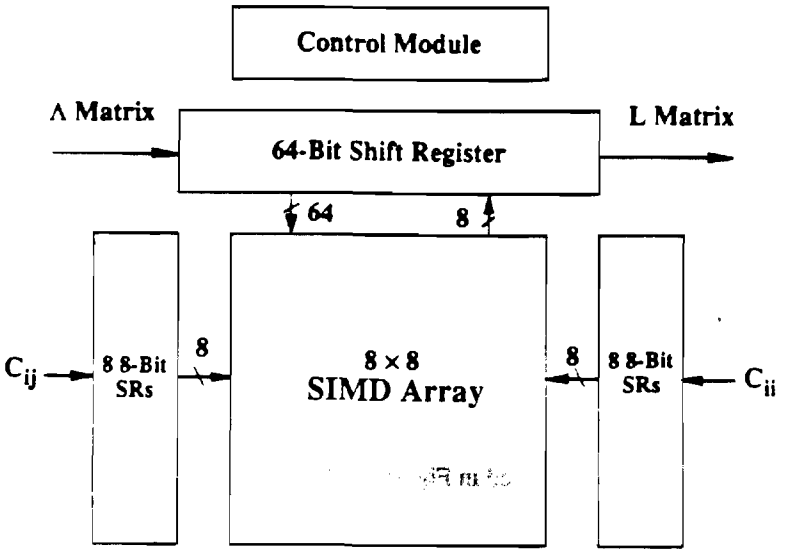


Figure 4: Circuit Block Diagram for DRA2

This diagram of four functional blocks is also served as the PPL layout floorplan for efficient layout (in section 3.5) and testing blocks to imbed the module testing strategy (in section 4).

2. Systolic Cells and Array Design

The basic principle of the systolic architecture for DRA is illustrated in Figure 5. By replacing a single Processing Element with an array of 8 by 8 PEs, a higher computation throughput can be achieved without increasing memory bandwidth. The function of the memory (i.e., the L matrix shift registers) in the diagram is to "pulse" data l_{jp} (j, p = 1,2,..., n) through the array of cells. Then new data l_{ik} (i, k = 1,2,..., n) are returned to memory in a rhythmic fashion. The **crux** of this approach is to ensure that once the data are brought out from the memory they can be used effectively at each cell they pass while being "pumped" from cell to cell along the array.

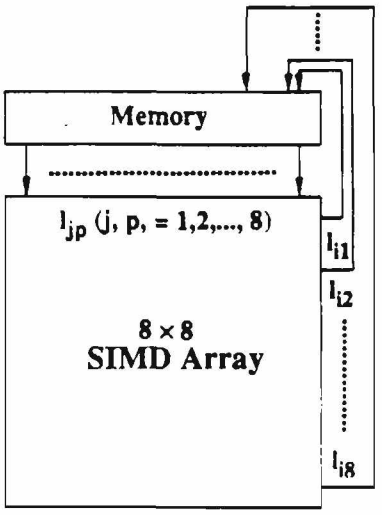
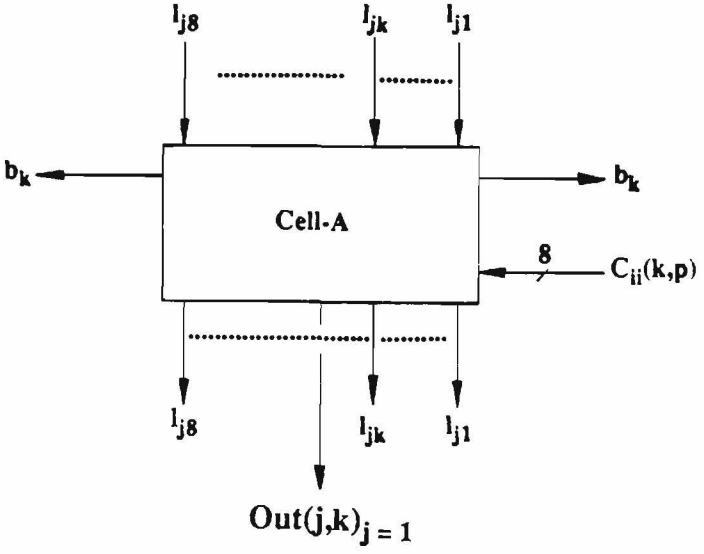


Figure 5: Basic Principle of the Systolic DRA2 System

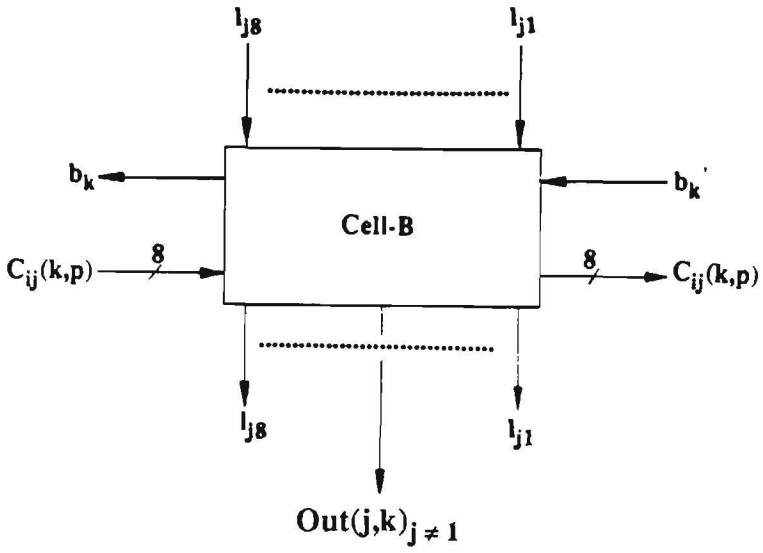
To perform parallel DRA computation, two cells (as illustrated in Figure 6 (a) and (b)) with almost identical logic and structure were used in constructing the systolic array. The only difference is that the first cell is in charge of generating broadcasting signals for each row array. The construction of the systolic array using these two cells is illustrated in Figure 7.



$b_k = l_{ik} = l_{jk}$ *at column* $j = 1$. (30)

$$Out(j,k)_{j=1} = \Sigma_{p=1}^{n} (l_{jp} \times \Lambda_{ij}(k,p)) = \Sigma_{p=1}^{n} (l_{jp} \times l_{ik} \times C_{ii}(k,p)) = \Sigma_{p=1}^{n} (l_{jp} \times b_k \times C_{ii}(k,p)) = \Sigma_{p=1}^{n} \overline{(\overline{l_{jp}} + \overline{b_k} + \overline{C_{ii}})}. \quad (31)$$

(a) Cell–A



$$b_{k(in)} = b_{k(out)}, \text{ at columns } j \neq 1. \quad (32)$$

$$Out(j,k)_{j=1} = \Sigma_{p=1}^{n} (l_{jp} \times \Lambda_{ij}(k,p)) = \Sigma_{p=1}^{n} (l_{jp} \times l_{ik} \times C_{ii}(k,p)) = \Sigma_{p=1}^{n} (l_{jp} \times b_k \times C_{ii}(k,p)) = \Sigma_{p=1}^{n} \overline{(\overline{l_{jp}} + \overline{b_k} + \overline{C_{ij}})}. \quad (33)$$

(b) Cell–B

Figure 6: Two Cells in DRA2 Systolic Array

According to Figure 3 and Eq. (30) to (33), these two cells can be implemented in two levels of NOR gate combinational logic. Their PPL layouts were shown in Figure 13.

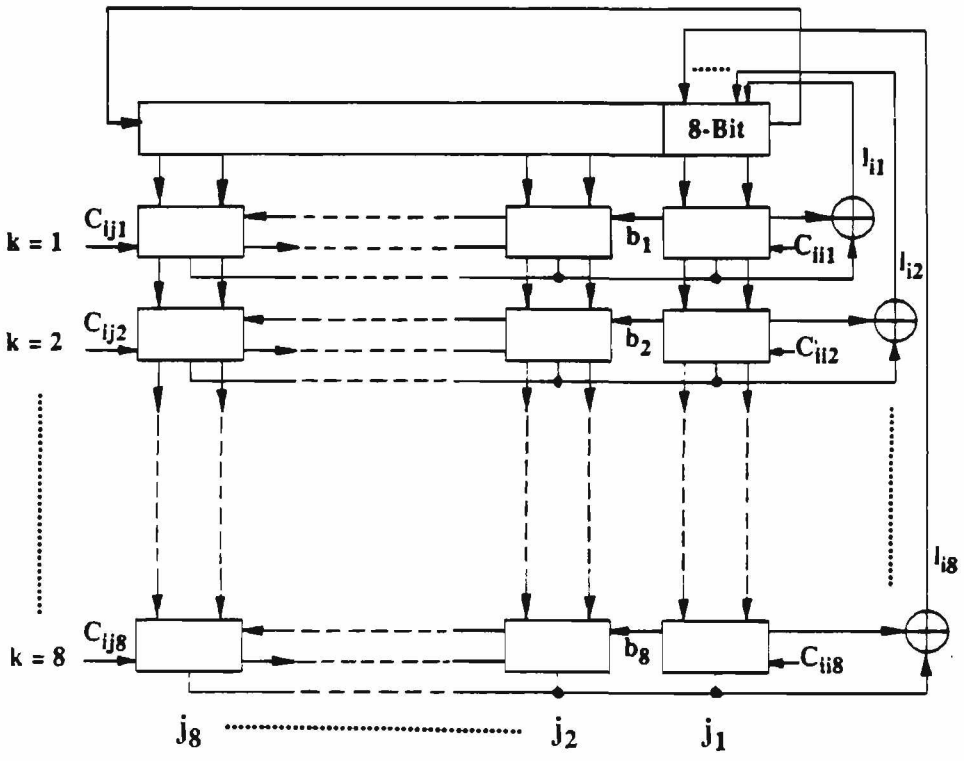


Figure 7: Construction of Systolic Array Using Cell-A and Cell-B

3. Circuit Features and Design Techniques

In addition to designing the simple and regular cells, several efficient techniques, such as interleaved processing, multiple signal broadcasting, and self-timed synchronization, were applied to the implementation of the systolic DRA2 architecture.

Recursive Systolic Computation and Interleaved Processing

Since the introduction of the time dimension in section 3, the systolic array in Figure 7 possesses a time-varying characteristic, which makes recursive systolic computation and interleaved processing possible. Let's focus on the first column (j = 1) array. It is clearly indicated that the first input vector, which is the i^{th} row vector of L, the labeling matrix, at the $n\text{-}1^{th}$ iteration, is fed into the first column of DRA2 array as

$$(l_{j1}, l_{j2}, l_{j3}, \ldots\ldots, l_{jn}),$$

where j = 1,...,n. The corresponding output vector L_i of the systolic array, which is the i^{th} row vector of L labeling matrix at the n^{th} iteration, is generated:

$$(l_{i1},l_{i2},l_{i3},.....,l_{in}),$$

where i is fixed at a time t=i. As time moves forward, the elements in the L shift register have shifted from the left to the right in an 8-clock-pace fashion. The time-varying feature of the entire DRA2 array can best be described by following two *Topological Index Equations*:

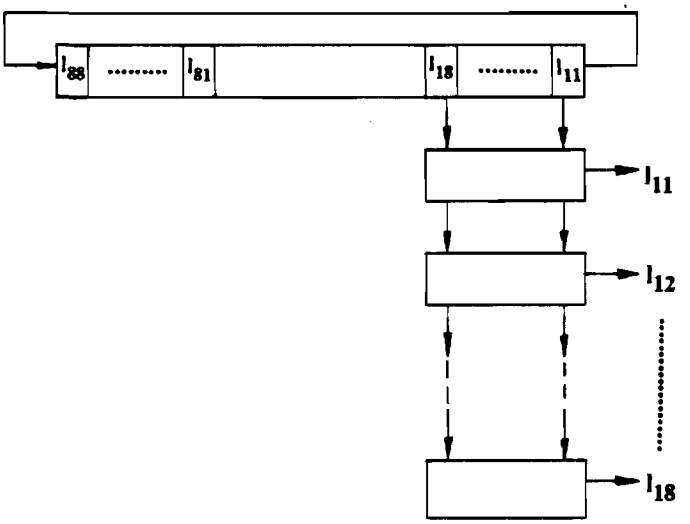
$$i \leftarrow i+t \bmod n \quad (34)$$

$$j \leftarrow j+t \bmod n \quad (35)$$

For example, in the DRA2 system, at time t = i = 1, vector L_1

$$(l_{11},l_{12},l_{13},l_{14},l_{15},l_{16},l_{17},l_{18})$$

is generated.



and at time t = i = 2, vector L_2

$$(l_{21},l_{22},l_{23},l_{24},l_{25},l_{26},l_{27},l_{28})$$

is generated, etc.

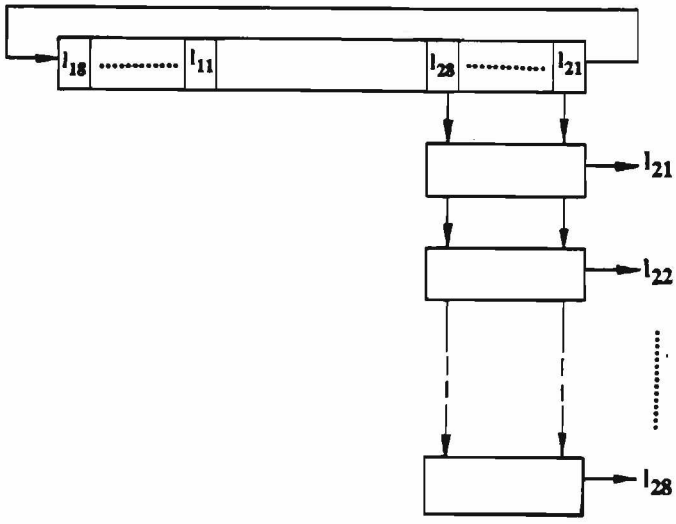


Figure 8: Computational Wavefront Pipelining and Circulation for Interleaved Processing

Each L_i vector is computed based on the interleaved utilization of the systolic array, whereas eight L_i vectors form an entire computational wavefront of the L labeling matrix, of the n^{th} relaxation iteration. Note that we use the number of n computing trees for generating n^2 l_{ik}'s in $O(n^2)$ time, we may also use n by n computing trees to compute the same number of l_{ik}'s in O(n) time, provided that the latter has a uniformly progressing wavefront in time and in space but the former doesn't.

Multiple Signal Broadcasting

The broadcasting technique is probably one of the most obvious ways to make multiple use of each input element. It plays an important role in making the parallel computation tree of Figure 3 implementable. Two multiple broadcasting schemes are used in DRA2 architecture. In the first, n^2 vertical broadcasting lines from each pipelining operand are connected to the bottom most leaves' node of each parallel computing tree. Secondly, as depicted in Figures 6 and 9, Cell-A at column j (=1) is used to jog signal $l_{ik}^{(n-1)}$ (which is the n^{th} l_{jp}) and then propagate it horizontally from right to left through the entire row array. Thus, the output vector of the systolic array, i.e., $(l_{i1},l_{i2},l_{i3},l_{i4},l_{i5},l_{i6},l_{i7},l_{i8})$, can be generated simultaneously in a highly concurrent manner. For the sake of simplifying the analysis in fast DRA3 and DRA4 architectures, we define the second multiple data routing pattern for jogging b_k as *J-Pattern* generation in Figure 9.

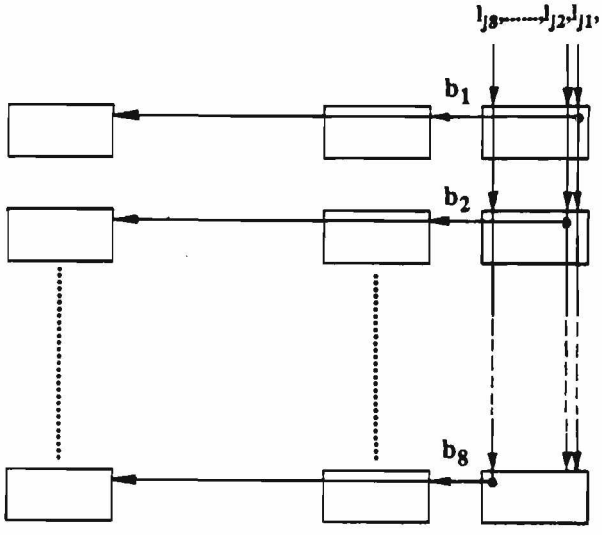


Figure 9: Broadcasting Scheme for b_k Signal

Self-Timed Synchronization and Tagged-Bit Control

By using recursive systolic computation and interleaved processing, the computational task has been decomposed into the smallest computing piece, L_i. To compute each vector L_i, the globally synchronized systolic array of Figure 7 is used. For completing the entire relaxation computation, this synchronous array is imbedded into a self-timed system. The self-timed asynchronous scheme may be costly in terms of extra hardware and delay in each element, but it has the advantage that the time required for a communication event between two elements is independent of the size of the entire system [14]. Also, it is easy to design and validate a self-timed state machine in PPL methodology [16].

Among the 64-bit L shift registers, the rightmost first 8-bit SR is one which is able to parallel load in the n^{th} output vector from the systolic array in order to update the current $n\text{-}1^{th}$ L_i row vector. This iteration and updating process is the core of the relaxation process described in Eq. (8). To sense the completion of computation, a Comparator is built on top of the first 8-bit SR. If two vectors are equal, a **row-eq** signal of 1 is produced and stored into **8-bit States SR** of the Control Module; otherwise a 0 signal is sent. As soon as the State Register gets eight 1's, which means the equality of Eq. (8) is reached, an **all-eq** signal is issued to the FSM. Since the control processes in this system are based on the data validity of a *control data flow*, a reliable and fast execution in a data-driven environment is created. The control mechanism used in the parallel DRA2 architecture is shown in Figure 10.

To ensure that the iteration cycle completes at the end of $n \times n$ cycles, a **tagged-bit** is derived from an ANDed term of both the l_{11} bit and the 64th-count of the Timer, which has served as a reliable alignment signal for computation in the control flow.

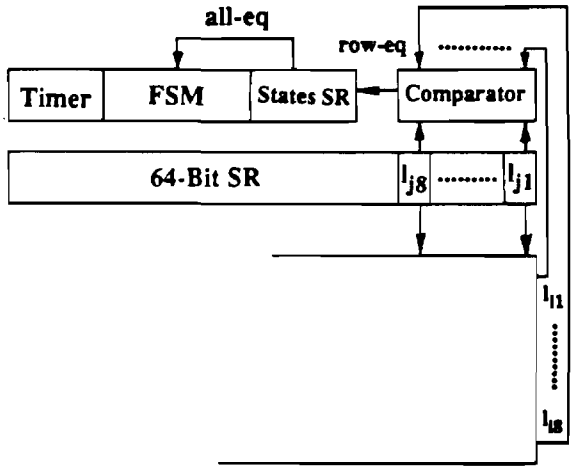


Figure 10: Self-Timed Synchronization

In Figure 11 the state graph of the FSM is illustrated.

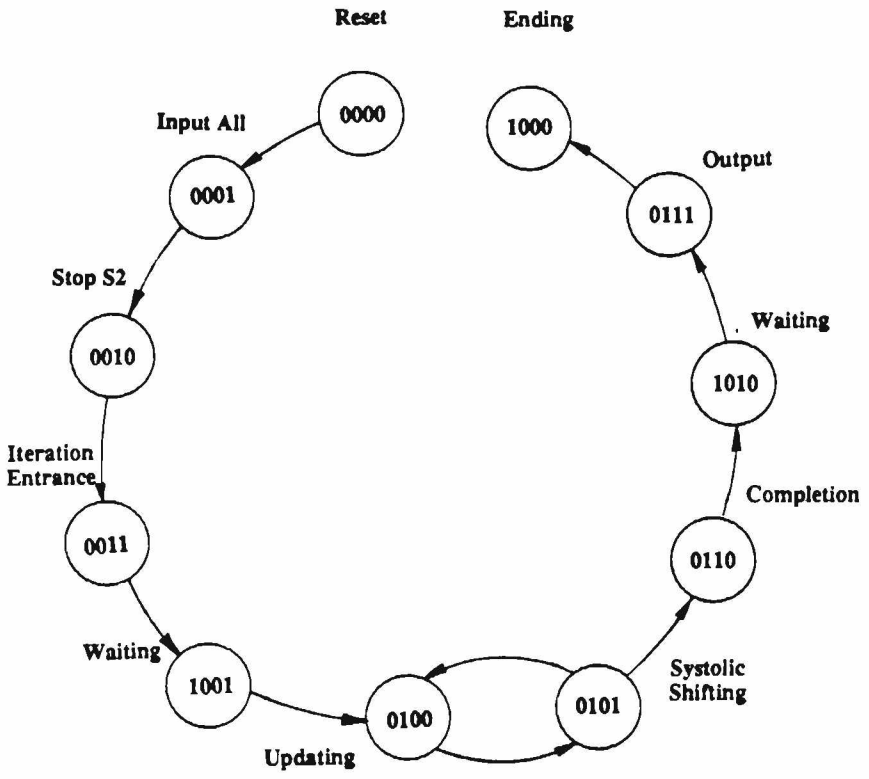


Figure 11: State Graph of the Finite State Machine

3.5 PPL Layout

The DRA2 chip was built by assembling the four functional blocks in Figure 4 using PPL (Path Programmable Logic) tools at the University of Utah [16,17]. Since parallel computation and the systolic array greatly simplify the design difficulties, the PPL layout is very simple and straightforward. An overview of the complete PPL layout, which is a PPL mapping of the block **floor plan** in Figure 4, is shown in Figure 12. PPL programs for Cell-A, Cell-B, Timer, Finite State Machine, State Register, and Comparator are shown in Figure 13. A new PPL layout scheme which is applied with a full-custom designed Cell-A and Cell-B is shown in Figure 14.

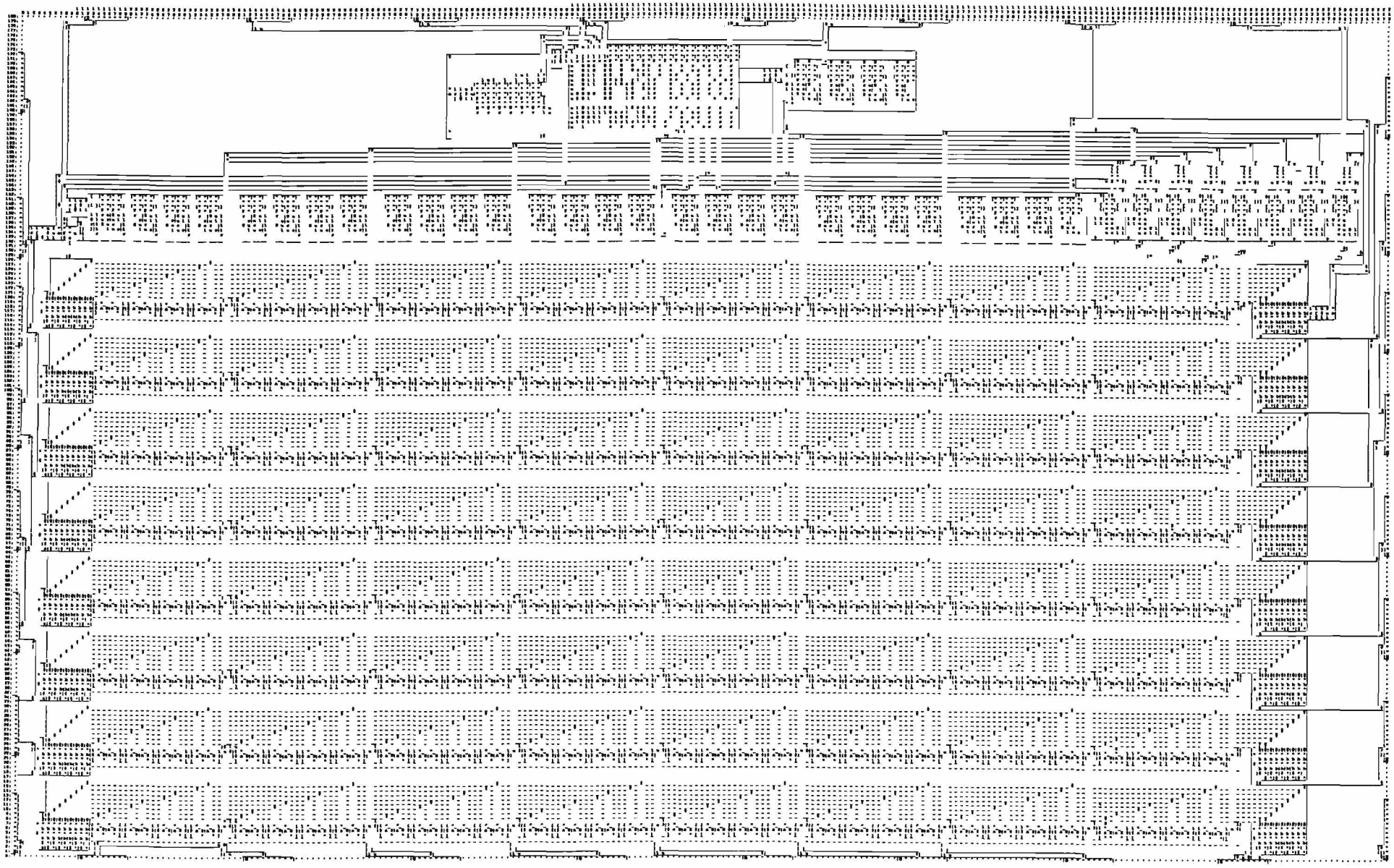


Figure 12: The PPL Layout of Discrete Relaxation Chip

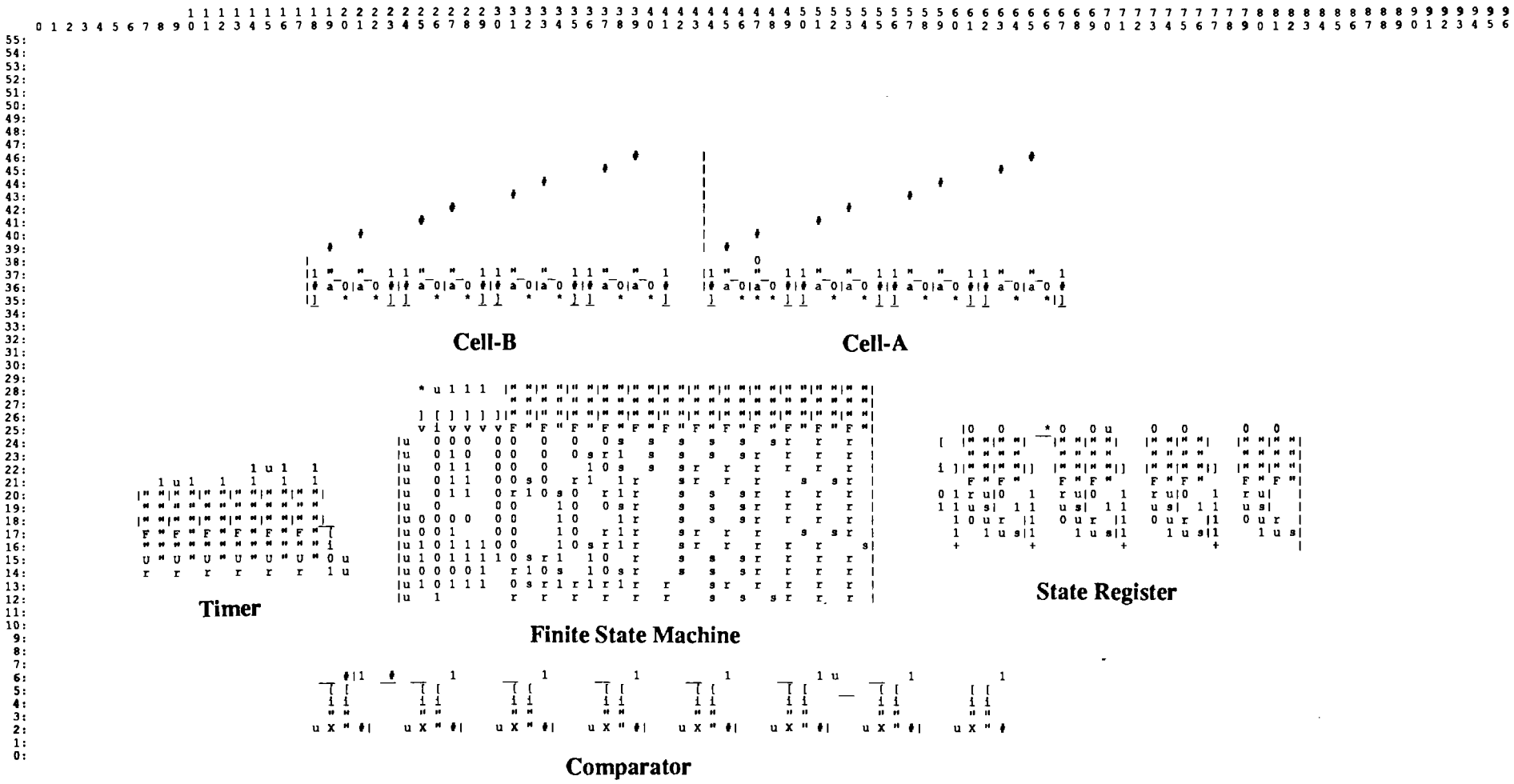


Figure 13: The PPL Layouts for Several Circuit Modules

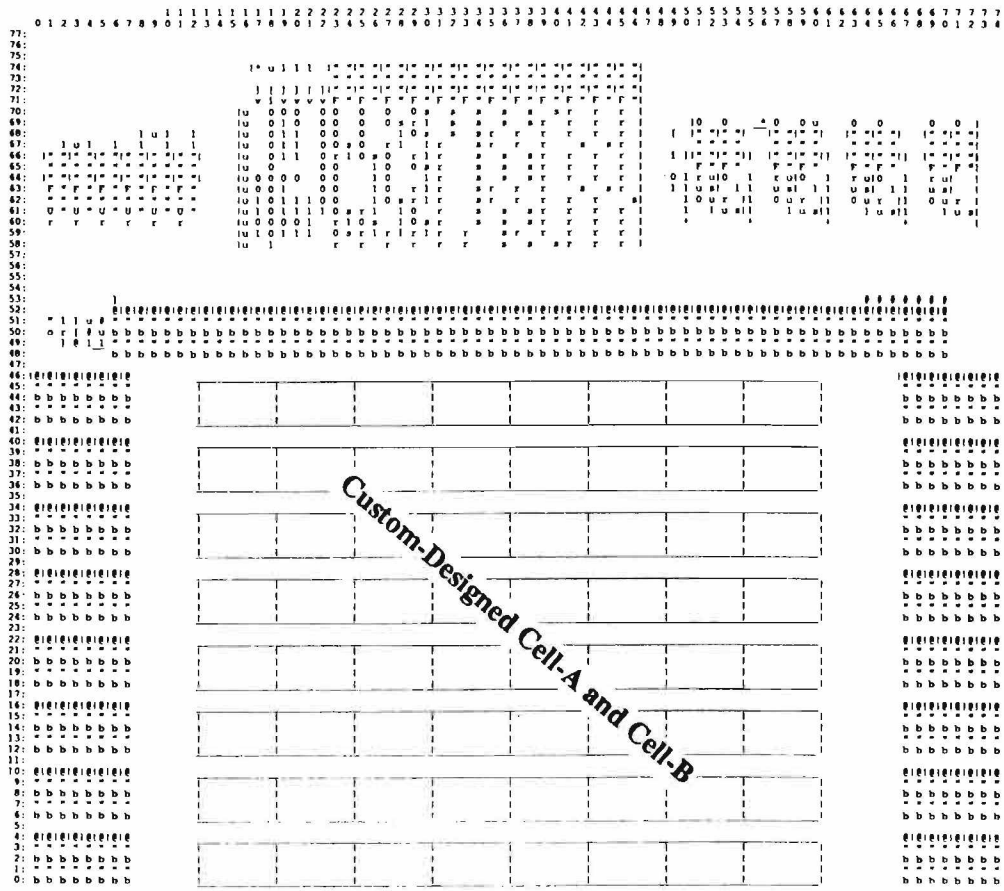


Figure 14: A New PPL layout Using Full-Custom Desinged Cell-A and Cell-B

3.6 Pin Descriptions and Interfacing with CPU

1. Pin descriptions

The pin out diagram is shown in Figure 15.

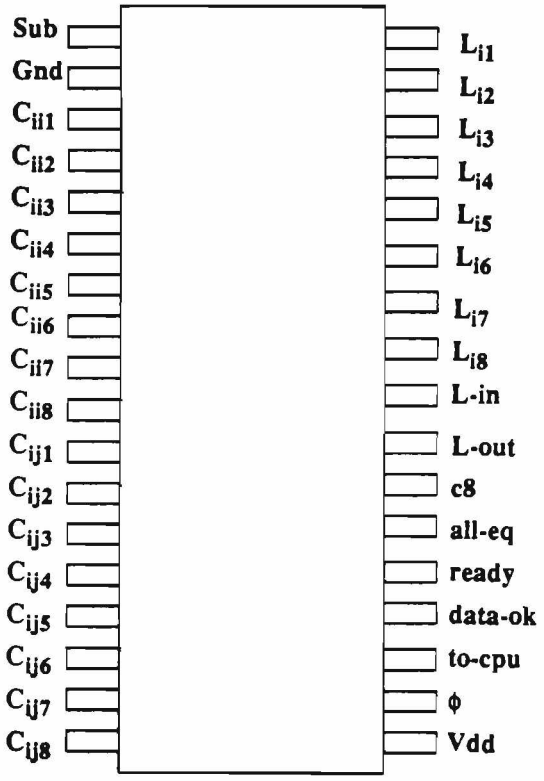


Figure 15: Pinout Diagram

A brief description of each pin and its function is outlined below:

1. $C_{ii1},...,C_{ii8}$: These 8 pins are used for data input and testing. The data of the C_{ii} elements can be shifted in from pins C_{ii1}, C_{ii3}, C_{ii5} and C_{ii7}, then the entire vector can be tested from all 8 pins.

2. $C_{ij1},...,C_{ij8}$: Same function as last 8 pins but they are for inputting matrix C_{ij} elements.

3. $L_{i1},...,L_{i8}$: There are a total of 8 pins for testing the output vector L_i of the systolic array.

4. **L-in** and **L-out:** Input and output pins for L matrix shift register.

5. **c8:** The 8-cycle flag for detecting systolic pace and control bit signal in testing mode. When every 8-clock is sensed, *c8* is set to 1.

6. **all-eq:** State signal for monitoring the ending of the iteration process. When all 1's of *all-eq* appears, the iteration is done.

7. **ready:** When relaxation computation is demanded and data (C_{ii}, C_{ij}, and $L^{(o)}$) are ready, CPU sets *ready* as 0, which resets all states of the Finite State Machine in the control module.

8. **data-ok:** Upon the completion of the relaxation computation, DRA1 issues *data-ok* by signaling a 1 to notify the CPU that the output of computation is pending for transfer.

9. **to-cpu:** Once the CPU finishes the preparation for receiving the data giving the result, a *to-cpu* command of 1's is acknowledged by the DRA1 chip, then a data transfer from DRA1 to CPU takes place.

10. ϕ: This clock is used to generate several 2-phases clocks with different signal polarity and delay.

11. **Vdd, Gnd:** For power supply.

2. Interfacing with CPU

Since the DRA1 chip is designed as a **Microcomputer Peripheral Device**, it can be easily interfaced with a CPU as illustrated in Figure 16. The data input pins of C_{ii}, C_{ij}, and **L-in**, **L-out** are tied to the data-bus. Pins like **ready**, **data-ok**, and **to-cpu** are connected to control pins of the CPU. Since **ready** is designed as the chip initialization signal it can be used as the Chip-Select signal preceded with a simple combinational circuit for a given address. To initiate the DRA1, the processor first places the data onto the input pins and selects/resets chip using **ready**, then DRA1 computes relaxation until **data-ok** is signalled to the CPU. Once the CPU is ready, data of the results can be read back to CPU after DRA1 senses **to-cpu** high.

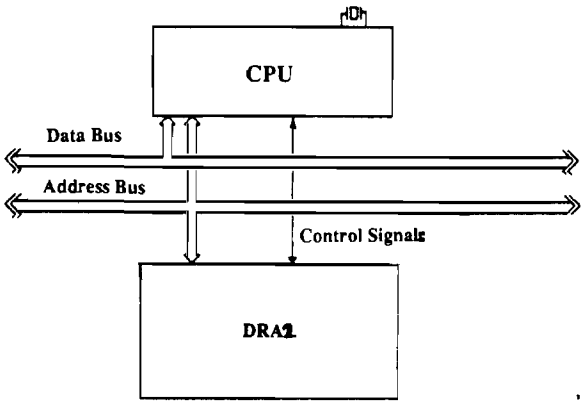


Figure 16: Interfacing Block Diagram

3.7 Simulation

1. Functional Simulation

Functional simulation is aimed at the verification of the correctness of the algorithm and data structure, discovery of limitations and problems which may occur during practical implementation. Thus a number of high-level functional simulations were performed during the formulation of DRA.

2. Logical Simulation

Logic simulation for the DRA1 chip was performed for individual modules and the entire circuit using the PPL topological circuit simulation tool **SIMPPL**. Simulation using SIMPPL is performed by assigning logical values to every node in the circuit. Input values are assigned and allowed to ripple through the circuit. Output values are then checked to ensure that the correct values are produced. For detailed functionality and usage about PPL simulation tools see [18].

An 8-label 8-object coloring identification problem was selected for logical simulation as shown in the following. The input to the circuit includes matrixes C_{ii}, C_{ij}, and $L^{(o)}$. Matrixes C_{ii} and C_{ij} are the inherent label pair's relationships. $L^{(o)}_{ij}$ are the raw data seen by a robot. The first seven initial labels (L_1,L_2,L_3,L_4,L_5,L_6,L_7) of $L^{(o)}$ indicate seven distinct regions of color on an object, the 8th label (L_8=11111111) means the color in the 8th region is a mixture of all 8 colors in these eight areas. We frequently meet such a situation. For example, suppose an airplane is flying, its major parts are clear, but one area in its body is blurred due to the plane's motion.

$$C_{ii}=\begin{pmatrix}1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&1\end{pmatrix} \tag{36}$$

$$C_{ij}=\begin{pmatrix}0&1&1&1&1&1&1&1\\1&0&1&1&1&1&1&1\\1&1&0&1&1&1&1&1\\1&1&1&0&1&1&1&1\\1&1&1&1&0&1&1&1\\1&1&1&1&1&0&1&1\\1&1&1&1&1&1&0&1\\1&1&1&1&1&1&1&0\end{pmatrix} \tag{37}$$

$$L^{(o)}=(L_1,L_2,L_3,L_4,L_5,L_6,L_7,L_8)^t=\begin{pmatrix}1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&0&0&1&0\\1&1&1&1&1&1&1&1\end{pmatrix}, \tag{38}$$

After relaxation computation, the 8th color in that region has been identified as L_8=00000001 in matrix $L^{(n)}$.The simulation file for these inputs is attached in Appendix, the final labeling matrix sought is:

$$L^{(n)}=(L_1,L_2,L_3,L_4,L_5,L_6,L_7,L_8)^t=\begin{pmatrix}1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&1\end{pmatrix}. \tag{39}$$

4. Testing

Since parallel reformulation and systolic array implementation minimize the major difficulties in the design of the DRA1 circuit, this results in a straightforward testing procedure. The testing will be performed on a Tektronix DAS 9100.

Two main testing strategies are: (1) Dividing the entire chip into several functional blocks as shown in Figure 4, testing each individual circuit inside the block for accurately locating the faults associated with the specific circuit

units; (2) Since DRA1 is designed as a microcomputer peripheral device, it can be tested in a microcomputer-based environment, as illustrated in Figure 16. These strategies can be imbedded in the following two steps.

4.1 Testing for Systolic Array

A total number of eight test pins ($L_{i1},...,L_{i8}$) were built in the chip for testing and monitoring the output vectors generated during each systolic cycle. These pins mathematically represent the n^{th} iteration results of the parallel computational trees. Since these results are predictable from high-level functional simulation or lower-level PPL logic simulation. Errors inside each combinational cell can be detected very easily. The testing of systolic array implies that shift registers for C_{ii}, C_{ij} and L need to be tested first. Testing for these registers have already been taken into account and will be carried out by using pins l-in, l-out, C_{ii1} to C_{ii8}, and C_{ij1} to C_{ij8}.

4.2 Testing for Iteration Process and Control Module

After the systolic array has passed the test, the DRA1 chip will be tested for the relaxation computation. During this stage only the iteration process of computation and control module need to be tested. In Figures 10 and 15 a number of control pins which provide sufficient iteration flags and control information are packaged on the chip, such as **c8**, **all-eq**, **ready**, **to-cpu**, and **data-ok**.

An advantage of designing the control module with a data-driven mechanism will be seen because of its convenience in testing. Referring to state diagram of FSM in Figure 11, the entire computation has been divided into several distinct periods with each period initialized based on the availability of certain critical control signals, for instance the most general indication signal for systolic shifting, updating, inputing, and outputting, the **c8**. Another nice side-effect is gained through the implementation of the interfacing pins with the CPU, which not only makes the testing available in a microcomputer-based environment, also separates testing procedure with interactive notification of testing states of these signals. Much of the time and effort saved during testing can be drawn from these advantages.

5.Comparison with A Conventional Design

5.1 A Brief Description of the DRA1 Design

A conventional design for an 8-object, 8-label DRA problem, called DRA1, is presented in [2]. The DRA1 system consists of three chips. A **DRA1 Chip** performs DRA computation. An **External RAM** is used for storing, prior to DRA computation, the matrices C_{ii}, C_{ij}, Λ_i, and Nei[i,j] elements. A **Bus Control Chip** coordinates the interaction between the DRA1 chip and the external RAM, under control of the host computer. The block diagram of the DRA1 chip is shown in Figure 17.

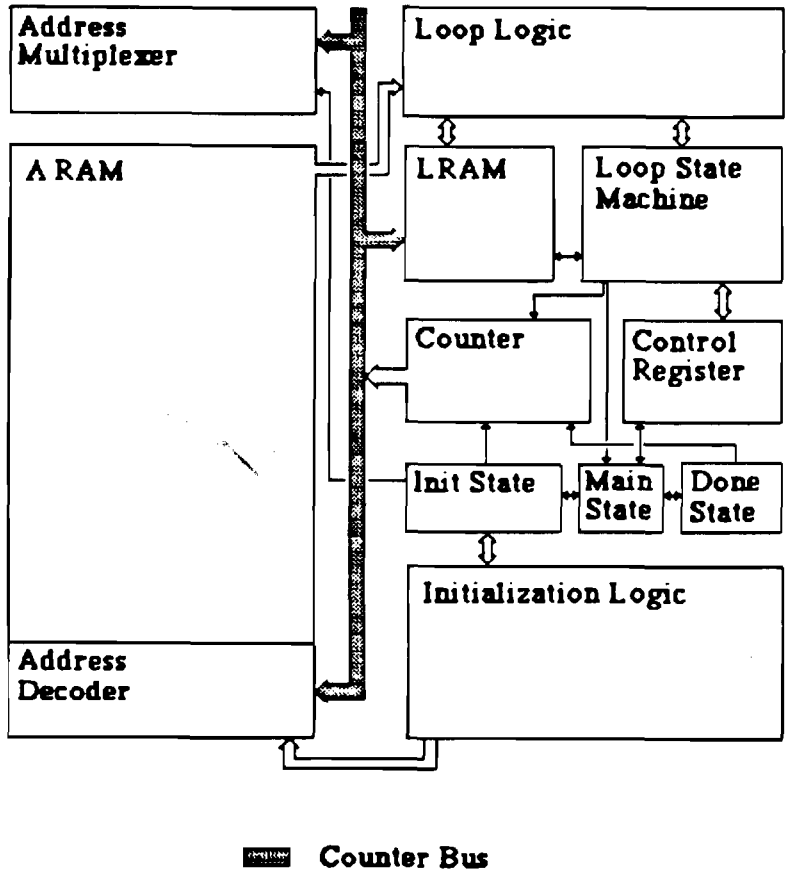


Figure 17: DRA1 Functional Block Diagram

These functional blocks are: (1) An **Internal RAM** for holding matrices $\Lambda_{ij}(k,p)$. (2) An n by n **Counter** for determining the number of the elements during data reloading. (3) An **Initialization Logic Unit** evaluates the matrices $\Lambda_{ij}(k,p)$ elements and writes them into RAM. (4) A **Loop Logic Unit** reads matrices $\Lambda_{ij}(k,p)$ elements

from RAM and iterates the labeling vectors L_is over these constraints, and writes the results into LRAM. (5) There are four **State Machines** which were built. The *Main State Machine* controls the three other state machines, activating each successively. The *Initialization State Machine* controls the initialization logic unit and the internal RAM, as indicated in Figure 18.

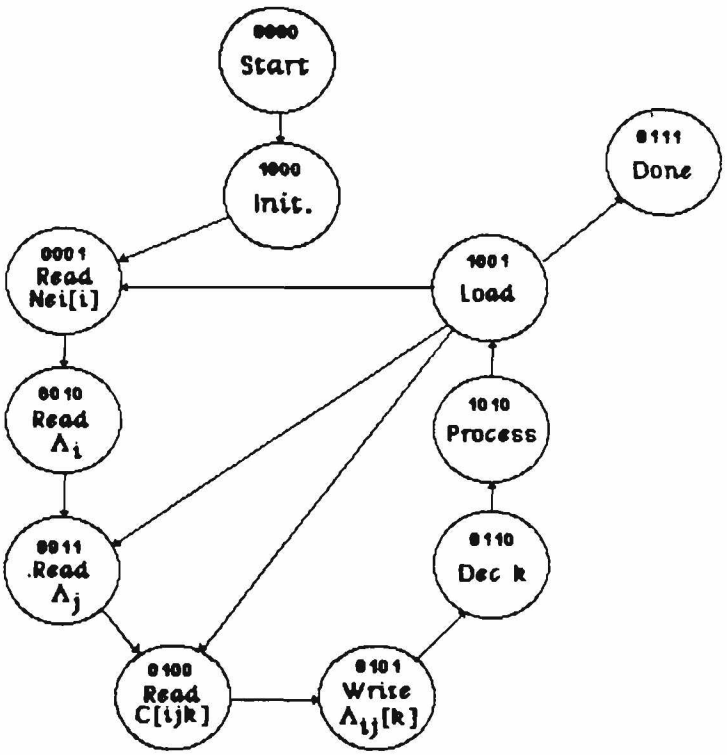


Figure 18: Initialization State Diagram

The *Loop State Machine* controls the loop logic unit and LRAM. The state diagram is shown in Figure 19.

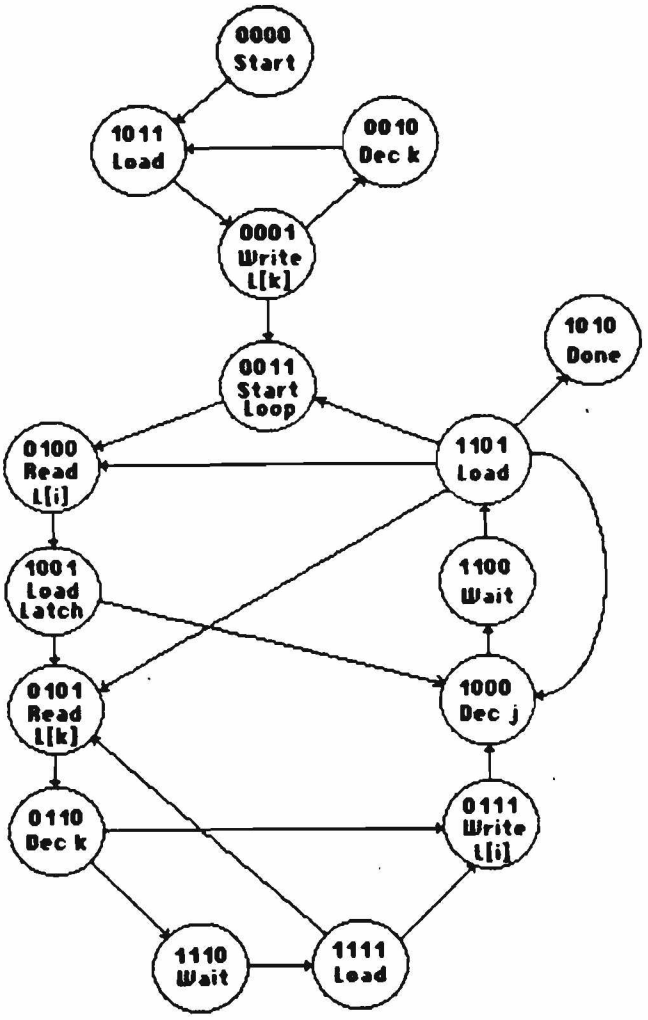


Figure 19: Loop State Diagram

The *Done State Machine* loads back the results upon the computation completion. The complete DRA1 chip was implemented as illustrated in Figures 20 and 21.

For time and space analysis of DRA1 circuit, see section 3.1 and [2].

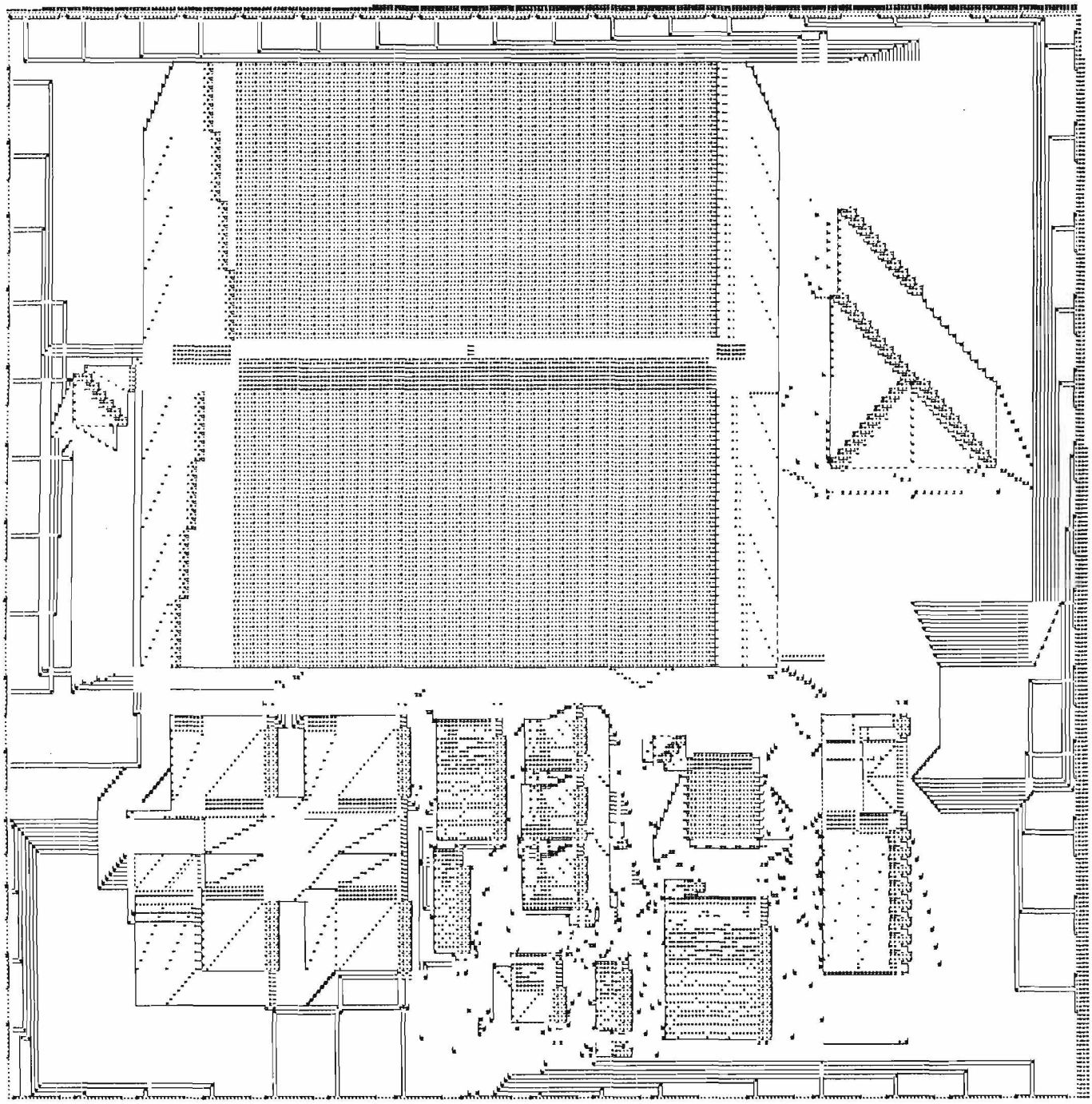


Figure 20: PPL Layout for A Conventional DRA1 Circuit

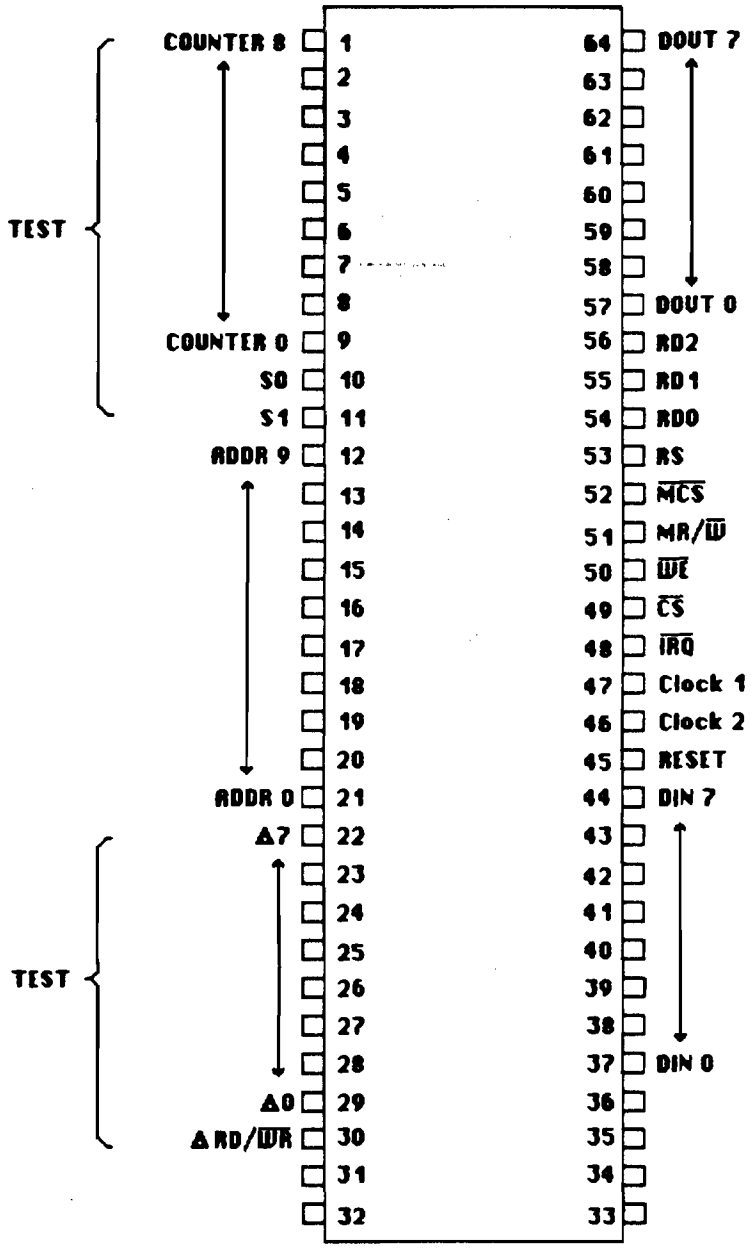


Figure 21: DRA1 Chip Pin Out Diagram

5.2 Comparisons

The conventional design DRA1 and DRA2 architectures are designed for the same DRA problem, a brief comparison between both designs are summarized in the following Figures.

Complexity Comparisons

	DRA2 Architecture	DRA1 System
Computational Family	Compute Bound	I/O Bound
Hardware Algorithm	Highly Concurrent	Serially Computing
Memory Requirement	None	12 K
Computing Time	Microseconds	Seconds to Minutes
Layout	Regular and Simple	Irregular and Hard
Control Strategy	Simple	Complicated
Entire System	One Chip	Three Chips (DRA1 chip plus External 4-K RAM and Bus Ctl Chip)
# of Transistors	6,382	≈ 80,000

Design and Cost Comparisons

	DRA2 Architecture	DRA1 System
Algorithm Reformulation	100 Hours	No
PPL Design Time	100 Hours	450 Hours
Computer Time (in $)	$300	$1500

DRA Chip Descriptions

	DRA2 Architecture	DRA1 System
# of Transistors	6,382	39,502
Pins	35	64
Sizes	181 × 249	300 × 250
(PPL row by column)	75 × 75	
	(Using Full-Custom Designed Cell-A and Cell-B in Systolic Array, it is currently under implementation)	

Figure 22: The Comparisons with A Conventional Design

6. Further Advanced Development

Further research in developing fast, high-performance discrete relaxation hardware algorithms and layout implementations is really attractive and promising. The major issues related to this research are:

1. Using full-custom designed PPL cells in DRA circuit design which is able to greatly decrease the layout area. Since just two cells need to be designed, almost no additional cost will be added.

2. The highest degree of flexibility in DRA design can be obtained by allowing programmability in cells as well as reconfigurability of cell interconnections. Thus to implement a *Programmable Systolic Chip* (PSC) [3] for more general or specific DRA problem seems necessary.

3. CMOS technology could be applied to give higher speed and lower power realization.

7.Conclusions

Several conclusions can be drawn from this extended summary:

1. The implementation of DRA has paved the way for developing various kinds of fast and high-performance **Discrete Relaxation Chip.**

2. PPL design tools are really a cost-effective and high-speed design tool, which frees the labor-intensive composite layout and many lower level logic design. Since each individual circuit unit can be layed out and simulated separately, the hierarchical VLSI design can be carried out efficiently. A weak point of this methodology is the space-consuming of its wire cells. Since every row wire and all wire connections take one cell space, a large amount of areas were wasted in wiring. In DRA circuit, for example, 80 percent area is spent on wire connection. To adopt PPL being suitable for parallel design in which generally a large numbers of parallel wiring are required, special wiring cells to perform area-efficient layout are important.

3. Design of highly parallel hardware algorithms is the key problem of logic design for fast, area-efficient, high-performance, and less-device VLSI systems. A considerable attention should be paied on hardware algorithms analyses before implementing a circuit [8,9,10].

8. Acknowledgement

We appreciate David C-L. Ku at Stanford University for his DRA1 system design.

REFERENCES

1. W. Wang and J. Gu, *An $O(n^2)$ Time Fast Discrete Relaxation Architecture*, Project Report, Department of Computer Science, University of Utah, March 1986.

2. D. Ku, *DRA1 Chip Implementation Report*, Project Report, Department of Computer Science, University of Utah, March 1986.

3. H. T. Kung, *Putting Inner Loops Automatically in Silicon*, Lecture Notes in Computer Science Vol. 163: VLSI Engineering, pp. 70-104, Edited by Tosiyasu L. Kunii, Springer-Verlag, 1985.

4. R. P. Brent and H. T. Kung, *The Area-Time Complexity of Binary Multiplication*, Journal of The ACM Vol. 28, No. 3. pp. 521-534, 1981.

5. H. T. Kung, *Why Systolic Architectures?* Computer Magazine 15(1):37-46, January, 1982.

6. C. E. Leiserson, *Area-Efficient VLSI Computation*, The MIT Press, 1983.

7. K. Hwang and F. A. briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill, 1984.

8. J. E. Savage and J. S. Vitter, *Parallelism in Space-Time Tradeoffs*, VLSI: Algorithms and Architectures, Edited by P. Bertolazzi and F. Luccio, North-Holland, 1985.

9. Z. Galil, *Optimal Parallel Algorithms - invited paper*, VLSI: Algorithms and Architectures, Edited by P. Bertolazzi and F. Luccio, North-Holland, 1985.

10. H. Yasuura and S. Yajima, *Hardware Algorithms for VLSI System*, Lecture Notes in Computer Science Vol. 163: VLSI Engineering, pp. 70-104, Edited by Tosiyasu L. Kunii, Springer-Verlag, 1985.

11. R. V. Southwell, *Relaxation Methods in Engineering Science*, Oxfo D. Waltz, *Understanding Line DraWings of Scenes with Shadows*, In Psychology of Computer Vision, Edited by P. H. Winston, pp. 19-91, McGraw-Hill, 1975.

12. T. C. Henderson and O. D. Faugeras, *Relaxation Techniques in Computer Vision*, Oxford University Press, London, To Appear.

13. P. H. Winston, *Artificial Intelligence*, Addison Wesley, 1984.

14. D. H. Ballard and C. M. Brown, *Computer Vision*, Prentice-Hall, Inc., 1982.

15. IEEE/ACM, *Proceedings of International Conference on Computer-Aided Design*, 1984 and 1985.

16. K. F. Smith, T. M. Carter and C. E. Hunt, *Structured Logic Design of Integrated Circuits Using the Storage/Logic Array (SLA)*, IEEE Trans. on Electron Devices, Vol. ED-29, No. 4, April 1982.

17. *PPL Manual*, VLSI Group, University of Utah.

APPENDIX

PPL Simulation File for Region Coloring Problem

```
>> clock phi:0100
>> ;
>> ;═══════════════════════════════════════════════════════════════════;
>> ;                                                                   ;
>> ;    PPL Logic Simulation for an 8-Label 8-Object Coloring Problem  ;
>> ;                                                                   ;
>> ;                          (DRA.out)                                ;
>> ;                                                                   ;
>> ;═══════════════════════════════════════════════════════════════════;
>> ;
>> ;
>> set ready:0

   ; As soon as CPU gets a ready = 0, it enables/resets DRA2 chip, and starts
   ; to input Cii (=rc) and Cij (=lc), and Lij (=l-in) elements.

>> cy
    1:4> l-out=X l-in=X Li=XXXXXXXX c8=X all-eq=X ready=0 data-ok=0 to-cpu=X
>> set ready:1 to-cpu:0
>> ;cy 1
>> set l-in:1
>> set  lc1:0 lc2:1 lc3:1 lc4:1 lc5:1 lc6:1 lc7:1 lc8:1
>> set  rc1:1 rc2:0 rc3:0 rc4:0 rc5:0 rc6:0 rc7:0 rc8:0
>> cy
    2:4> l-out=X l-in=1 Li=XXXXXXXX c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 2
>> set l-in:0
>> set  lc1:1 lc2:0 lc3:1 lc4:1 lc5:1 lc6:1 lc7:1 lc8:1
>> set  rc1:0 rc2:1 rc3:0 rc4:0 rc5:0 rc6:0 rc7:0 rc8:0
>> cy
    3:4> l-out=X l-in=0 Li=XXXXXXXX c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 3
>> set l-in:0
>> set  lc1:1 lc2:1 lc3:0 lc4:1 lc5:1 lc6:1 lc7:1 lc8:1
>> set  rc1:0 rc2:0 rc3:1 rc4:0 rc5:0 rc6:0 rc7:0 rc8:0
>> cy
    4:4> l-out=X l-in=0 Li=XXXXXXXX c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 4
>> set l-in:0
>> set  lc1:1 lc2:1 lc3:1 lc4:0 lc5:1 lc6:1 lc7:1 lc8:1
>> set  rc1:0 rc2:0 rc3:0 rc4:1 rc5:0 rc6:0 rc7:0 rc8:0
>> cy
    5:4> l-out=X l-in=0 Li=XXXXXXXX c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 5
>> set l-in:0
>> set  lc1:1 lc2:1 lc3:1 lc4:1 lc5:0 lc6:1 lc7:1 lc8:1
```

```
>> set  rc1:0 rc2:0 rc3:0 rc4:0 rc5:1 rc6:0 rc7:0 rc8:0
>> cy
    6:4> l-out=X l-in=0 Li=XXXXXXXX c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 6
>> set l-in:0
>> set  lc1:1 lc2:1 lc3:1 lc4:1 lc5:1 lc6:0 lc7:1 lc8:1
>> set  rc1:0 rc2:0 rc3:0 rc4:0 rc5:0 rc6:1 rc7:0 rc8:0
>> cy
    7:4> l-out=X l-in=0 Li=XXXXXXXX c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 7
>> set l-in:0
>> set  lc1:1 lc2:1 lc3:1 lc4:1 lc5:1 lc6:1 lc7:0 lc8:1
>> set  rc1:0 rc2:0 rc3:0 rc4:0 rc5:0 rc6:0 rc7:1 rc8:0
>> cy
    8:4> l-out=X l-in=0 Li=XXXXXXXX c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 8
>> set l-in:0
>> set  lc1:1 lc2:1 lc3:1 lc4:1 lc5:1 lc6:1 lc7:1 lc8:0
>> set  rc1:0 rc2:0 rc3:0 rc4:0 rc5:0 rc6:0 rc7:0 rc8:1
>> cy
    9:4> l-out=X l-in=0 Li=0XXXXXXX c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 9
>> set l-in:0
>> cy
   10:4> l-out=X l-in=0 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 10
>> set l-in:1
>> cy
   11:4> l-out=X l-in=1 Li=XXXXXXX0 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 11
>> set l-in:0
>> cy
   12:4> l-out=X l-in=0 Li=XXXXXX0X c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 12
>> set l-in:0
>> cy
   13:4> l-out=X l-in=0 Li=XXXXX0XX c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 13
>> set l-in:0
>> cy
   14:4> l-out=X l-in=0 Li=XXXX0XXX c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 14
>> set l-in:0
>> cy
   15:4> l-out=X l-in=0 Li=XXX0XXXX c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 15
>> set l-in:0
```

```
>> cy
   16:4> l-out=X l-in=0 Li=XX0XXXXX c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 16
>> set l-in:0
>> cy
   17:4> l-out=X l-in=0 Li=00XXXXXX c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 17
>> set l-in:0
>> cy
   18:4> l-out=X l-in=0 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 18
>> set l-in:0
>> cy
   19:4> l-out=X l-in=0 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 19
>> set l-in:1
>> cy
   20:4> l-out=X l-in=1 Li=XXXXXX00 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 20
>> set l-in:0
>> cy
   21:4> l-out=X l-in=0 Li=XXXXX00X c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 21
>> set l-in:0
>> cy
   22:4> l-out=X l-in=0 Li=XXXX00XX c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 22
>> set l-in:0
>> cy
   23:4> l-out=X l-in=0 Li=XXX00XXX c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 23
>> set l-in:0
>> cy
   24:4> l-out=X l-in=0 Li=XX00XXXX c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 24
>> set l-in:0
>> cy
   25:4> l-out=X l-in=0 Li=000XXXXX c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 25
>> set l-in:0
>> cy
   26:4> l-out=X l-in=0 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 26
>> set l-in:0
>> cy
   27:4> l-out=X l-in=0 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
```

```
>> ;
>> ;cy 27
>> set l-in:0
>> cy
   28:4> l-out=X l-in=0 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 28
>> set l-in:1
>> cy
   29:4> l-out=X l-in=1 Li=XXXXX000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 29
>> set l-in:0
>> cy
   30:4> l-out=X l-in=0 Li=XXXX000X c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 30
>> set l-in:0
>> cy
   31:4> l-out=X l-in=0 Li=XXX000XX c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 31
>> set l-in:0
>> cy
   32:4> l-out=X l-in=0 Li=XX000XXX c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 32
>> set l-in:0
>> cy
   33:4> l-out=X l-in=0 Li=0000XXXX c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 33
>> set l-in:0
>> cy
   34:4> l-out=X l-in=0 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 34
>> set l-in:0
>> cy
   35:4> l-out=X l-in=0 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 35
>> set l-in:0
>> cy
   36:4> l-out=X l-in=0 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 36
>> set l-in:0
>> cy
   37:4> l-out=X l-in=0 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 37
>> set l-in:1
>> cy
   38:4> l-out=X l-in=1 Li=XXXX0000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 38
```

```
>> set l-in:0
>> cy
   39:4> l-out=X l-in=0 Li=XXX0000X c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 39
>> set l-in:0
>> cy
   40:4> l-out=X l-in=0 Li=XX0000XX c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 40
>> set l-in:0
>> cy
   41:4> l-out=X l-in=0 Li=00000XXX c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 41
>> set l-in:0
>> cy
   42:4> l-out=X l-in=0 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 42
>> set l-in:0
>> cy
   43:4> l-out=X l-in=0 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 43
>> set l-in:0
>> cy
   44:4> l-out=X l-in=0 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 44
>> set l-in:0
>> cy
   45:4> l-out=X l-in=0 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 45
>> set l-in:0
>> cy
   46:4> l-out=X l-in=0 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 46
>> set l-in:1
>> cy
   47:4> l-out=X l-in=1 Li=XXX00000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 47
>> set l-in:0
>> cy
   48:4> l-out=X l-in=0 Li=XX00000X c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 48
>> set l-in:0
>> cy
   49:4> l-out=X l-in=0 Li=000000XX c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 49
>> set l-in:0
>> cy
```

```
   50:4> l-out=X l-in=0 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 50
>> set l-in:0
>> cy
   51:4> l-out=X l-in=0 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 51
>> set l-in:0
>> cy
   52:4> l-out=X l-in=0 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 52
>> set l-in:0
>> cy
   53:4> l-out=X l-in=0 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 53
>> set l-in:0
>> cy
   54:4> l-out=X l-in=0 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 54
>> set l-in:0
>> cy
   55:4> l-out=X l-in=0 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 55
>> set l-in:1
>> cy
   56:4> l-out=X l-in=1 Li=XX000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 56
>> set l-in:0
>> cy
   57:4> l-out=X l-in=0 Li=0000000X c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 57
>> set l-in:1
>> cy
   58:4> l-out=X l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 58
>> set l-in:1
>> cy
   59:4> l-out=X l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 59
>> set l-in:1
>> cy
   60:4> l-out=X l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 60
>> set l-in:1
>> cy
   61:4> l-out=X l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
```

```
>> ;cy 61
>> set l-in:1
>> cy
   62:4> l-out=X l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 62
>> set l-in:1
>> cy
   63:4> l-out=X l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 63
>> set l-in:1
>> cy
   64:4> l-out=X l-in=1 Li=X1000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
>> ;
>> ;cy 64
>> set l-in:1
>> cy 150

   ; A row vector L1 is generated at cy 65 after inputing necessary data.
   ; Since c8 gets 1, FSM stops counting, in that c8 remains 1, and the
   ; computation enters iteration entrance at cy 65. The new vector updates
   ; the current L1 vector at cy 66.

   65:4> l-out=1 l-in=1 Li=10000000 c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
   66:4> l-out=1 l-in=1 Li=10000000 c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
   67:4> l-out=1 l-in=1 Li=10000000 c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
   68:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
   69:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
   70:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
   71:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
   72:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
   73:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
   74:4> l-out=0 l-in=1 Li=00100000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
   75:4> l-out=0 l-in=1 Li=01000000 c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
   76:4> l-out=0 l-in=1 Li=01000000 c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0

   ; Same situation as above, once c8 gets an 1, FSM stops (at cy 75) counting
   ; of c8 and updating current vector L2 (at cy 76). The relaxation iteration
   ; keeps going on in such way as shown in the fillowing file.

   77:4> l-out=1 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
   78:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
   79:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
   80:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
   81:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
   82:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
   83:4> l-out=0 l-in=1 Li=00010000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
   84:4> l-out=0 l-in=1 Li=00100000 c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
   85:4> l-out=0 l-in=1 Li=00100000 c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
   86:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
   87:4> l-out=1 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
   88:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
   89:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
   90:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
   91:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
   92:4> l-out=0 l-in=1 Li=00001000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
```

```
 93:4> l-out=0 l-in=1 Li=00010000 c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
 94:4> l-out=0 l-in=1 Li=00010000 c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
 95:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
 96:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
 97:4> l-out=1 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
 98:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
 99:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
100:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
101:4> l-out=0 l-in=1 Li=00000100 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
102:4> l-out=0 l-in=1 Li=00001000 c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
103:4> l-out=0 l-in=1 Li=00001000 c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
104:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
105:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
106:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
107:4> l-out=1 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
108:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
109:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
110:4> l-out=0 l-in=1 Li=00000010 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
111:4> l-out=0 l-in=1 Li=00000100 c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
112:4> l-out=0 l-in=1 Li=00000100 c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
113:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
114:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
115:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
116:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
117:4> l-out=1 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
118:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
119:4> l-out=0 l-in=1 Li=00000001 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
120:4> l-out=0 l-in=1 Li=00000010 c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
121:4> l-out=0 l-in=1 Li=00000010 c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
122:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
123:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
124:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
125:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
126:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
127:4> l-out=1 l-in=1 Li=00000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
128:4> l-out=0 l-in=1 Li=01000000 c8=0 all-eq=X ready=1 data-ok=0 to-cpu=0
129:4> l-out=1 l-in=1 Li=00000001 c8=1 all-eq=X ready=1 data-ok=0 to-cpu=0
130:4> l-out=0 l-in=1 Li=00000001 c8=1 all-eq=0 ready=1 data-ok=0 to-cpu=0

 ; The all-eq signal is set to 0 since 8-bit state register have loaded in
 ; to 8 definite data. The iteration still keeps going on for an all-eq = 1
 ; is appeared.

131:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
132:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
133:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
134:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
135:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
136:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
137:4> l-out=1 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
138:4> l-out=1 l-in=1 Li=10000000 c8=1 all-eq=0 ready=1 data-ok=0 to-cpu=0
139:4> l-out=1 l-in=1 Li=10000000 c8=1 all-eq=0 ready=1 data-ok=0 to-cpu=0
140:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
141:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
142:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
143:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
144:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
```

```
145:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
146:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
147:4> l-out=0 l-in=1 Li=01000000 c8=1 all-eq=0 ready=1 data-ok=0 to-cpu=0
148:4> l-out=0 l-in=1 Li=01000000 c8=1 all-eq=0 ready=1 data-ok=0 to-cpu=0
149:4> l-out=1 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
150:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
151:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
152:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
153:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
154:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
155:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
156:4> l-out=0 l-in=1 Li=00100000 c8=1 all-eq=0 ready=1 data-ok=0 to-cpu=0
157:4> l-out=0 l-in=1 Li=00100000 c8=1 all-eq=0 ready=1 data-ok=0 to-cpu=0
158:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
159:4> l-out=1 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
160:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
161:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
162:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
163:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
164:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
165:4> l-out=0 l-in=1 Li=00010000 c8=1 all-eq=0 ready=1 data-ok=0 to-cpu=0
166:4> l-out=0 l-in=1 Li=00010000 c8=1 all-eq=0 ready=1 data-ok=0 to-cpu=0
167:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
168:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
169:4> l-out=1 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
170:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
171:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
172:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
173:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
174:4> l-out=0 l-in=1 Li=00001000 c8=1 all-eq=0 ready=1 data-ok=0 to-cpu=0
175:4> l-out=0 l-in=1 Li=00001000 c8=1 all-eq=0 ready=1 data-ok=0 to-cpu=0
176:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
177:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
178:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
179:4> l-out=1 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
180:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
181:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
182:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
183:4> l-out=0 l-in=1 Li=00000100 c8=1 all-eq=0 ready=1 data-ok=0 to-cpu=0
184:4> l-out=0 l-in=1 Li=00000100 c8=1 all-eq=0 ready=1 data-ok=0 to-cpu=0
185:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
186:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
187:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
188:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
189:4> l-out=1 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
190:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
191:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
192:4> l-out=0 l-in=1 Li=00000010 c8=1 all-eq=0 ready=1 data-ok=0 to-cpu=0
193:4> l-out=0 l-in=1 Li=00000010 c8=1 all-eq=0 ready=1 data-ok=0 to-cpu=0
194:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
195:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
196:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
197:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
198:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
199:4> l-out=1 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
200:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=0 ready=1 data-ok=0 to-cpu=0
201:4> l-out=0 l-in=1 Li=00000001 c8=1 all-eq=0 ready=1 data-ok=0 to-cpu=0
```

```
  202:4> l-out=0 l-in=1 Li=00000001 c8=1 all-eq=1 ready=1 data-ok=0 to-cpu=0
  203:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=0
  204:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=0
  205:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=0
  206:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=0
  207:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=0
  208:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=0
  209:4> l-out=1 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=0
  210:4> l-out=1 l-in=1 Li=10000000 c8=1 all-eq=1 ready=1 data-ok=1 to-cpu=0

   ; Ok, now an 1 of all-eq is reached at cy 210. DRA2 sends a data-ok to CPU
   ; and then waits for a signal from CPU.

  211:4> l-out=1 l-in=1 Li=10000000 c8=1 all-eq=1 ready=1 data-ok=1 to-cpu=0
  212:4> l-out=1 l-in=1 Li=10000000 c8=1 all-eq=1 ready=1 data-ok=1 to-cpu=0
  213:4> l-out=1 l-in=1 Li=10000000 c8=1 all-eq=1 ready=1 data-ok=1 to-cpu=0
  214:4> l-out=1 l-in=1 Li=10000000 c8=1 all-eq=1 ready=1 data-ok=1 to-cpu=0
>>
sim> set to-cpu:1

   ; A to-cpu is sent to DAR2 which initiates the data transfering from DRA2
   ; to CPU. The following eight 8 cycles are the process to output the 8
   ; row vectors of L matrix. Look at l-out:

sim> cy 8
  215:4> l-out=1 l-in=1 Li=10000000 c8=1 all-eq=1 ready=1 data-ok=0 to-cpu=1
  216:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  217:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  218:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  219:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  220:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  221:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  222:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1

   ; This result means that the first row vector of L matrix is L1=10000000.

sim> cy 8
  223:4> l-out=0 l-in=1 Li=01000000 c8=1 all-eq=1 ready=1 data-ok=0 to-cpu=1
  224:4> l-out=1 l-in=1 Li=10000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  225:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  226:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  227:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  228:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  229:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  230:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1

   ; The second row vector is L2=01000000.

sim> cy 8
  231:4> l-out=0 l-in=1 Li=00100000 c8=1 all-eq=1 ready=1 data-ok=0 to-cpu=1
  232:4> l-out=0 l-in=1 Li=01000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  233:4> l-out=1 l-in=1 Li=10000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  234:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  235:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  236:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  237:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  238:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
```

```
   ; The third row vector is L3=00100000.

sim> cy 8
  239:4> l-out=0 l-in=1 Li=00010000 c8=1 all-eq=1 ready=1 data-ok=0 to-cpu=1
  240:4> l-out=0 l-in=1 Li=00100000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  241:4> l-out=0 l-in=1 Li=01000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  242:4> l-out=1 l-in=1 Li=10000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  243:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  244:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  245:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  246:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1

   ; L4=00010000.

sim> cy 8
  247:4> l-out=0 l-in=1 Li=00001000 c8=1 all-eq=1 ready=1 data-ok=0 to-cpu=1
  248:4> l-out=0 l-in=1 Li=00010000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  249:4> l-out=0 l-in=1 Li=00100000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  250:4> l-out=0 l-in=1 Li=01000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  251:4> l-out=1 l-in=1 Li=10000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  252:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  253:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  254:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1

   ; L5=00001000.

sim> cy 8
  255:4> l-out=0 l-in=1 Li=00000100 c8=1 all-eq=1 ready=1 data-ok=0 to-cpu=1
  256:4> l-out=0 l-in=1 Li=00001000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  257:4> l-out=0 l-in=1 Li=00010000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  258:4> l-out=0 l-in=1 Li=00100000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  259:4> l-out=0 l-in=1 Li=01000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  260:4> l-out=1 l-in=1 Li=10000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  261:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  262:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1

   ; L6=00000100.

sim> cy 8
  263:4> l-out=0 l-in=1 Li=00000010 c8=1 all-eq=1 ready=1 data-ok=0 to-cpu=1
  264:4> l-out=0 l-in=1 Li=00000100 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  265:4> l-out=0 l-in=1 Li=00001000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  266:4> l-out=0 l-in=1 Li=00010000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  267:4> l-out=0 l-in=1 Li=00100000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  268:4> l-out=0 l-in=1 Li=01000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  269:4> l-out=1 l-in=1 Li=10000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  270:4> l-out=0 l-in=1 Li=00000000 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1

   ; L7=00000010.

sim> cy 8
  271:4> l-out=0 l-in=1 Li=00000001 c8=1 all-eq=1 ready=1 data-ok=0 to-cpu=1
  272:4> l-out=0 l-in=1 Li=00000011 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  273:4> l-out=0 l-in=1 Li=00000111 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  274:4> l-out=0 l-in=1 Li=00001111 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  275:4> l-out=0 l-in=1 Li=00011111 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  276:4> l-out=0 l-in=1 Li=00111111 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
```

```
  277:4> l-out=0 l-in=1 Li=01111111 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1
  278:4> l-out=1 l-in=1 Li=11111111 c8=0 all-eq=1 ready=1 data-ok=0 to-cpu=1

   ; L8=00000001.

sim> q
```